

マイクロ
コンピュータ

インターフェースの作り方

大川善邦著

《電子情報技術》
75



産報出版

トップをやさしくおくる

《電子科学シリーズ》は、大学および工業高校卒業程度の方々に、新しい電子技術各分野の基礎知識とその実際を、やさしく豊富に解説するものです。

正副テキストに自習書に、各現場で広く利用され好評をいただいております。専門外関連領域を理解される上にも好適な、わが国唯一のシリーズです。

本書の特長

- インターフェースを作る基礎として、データベースや割込み処理から説明し、製作例にいたるまで、I/O中心に解説した。
 - 中心となるプロセッサは8080A、8085を取り上げてある。
 - アプリケーションも説明してあるので、マイクロコンピュータを応用する際の示唆が得られる。
-



マイク
コンピュ
タロ

インターフェースの作り方

大川善邦著

《電子科学シリーズ》

75



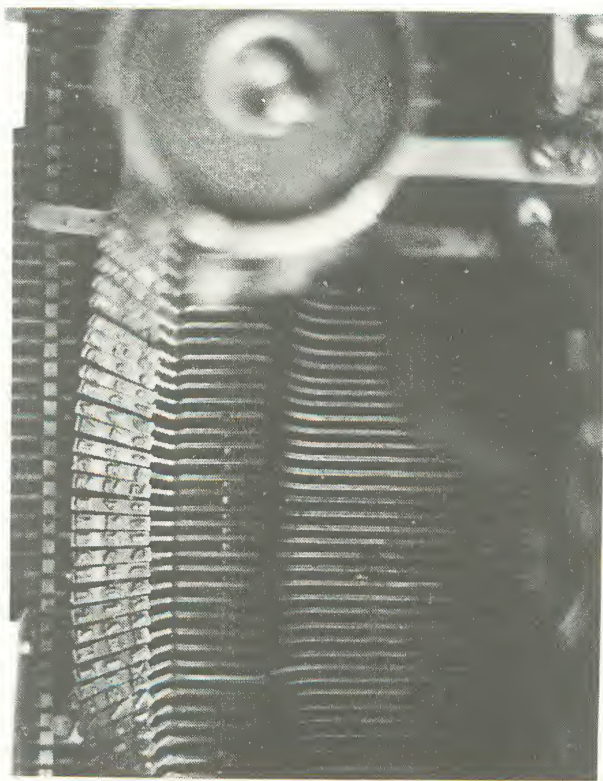


《電子科学シリーズ》75

マイクロ
コンピュータ

インターフェースの作り方

大川善邦 著



産報出版

ま え が き

最近、マイクロコンピュータに対する関心は、おそろしいほどの勢いで盛り上がってきています。3年ばかり前に私は、ある会議の席上で調子をつけるために、マイクロコンピュータの人口は将来百万人に近づくかもしれない、といったことがあります。当時、誰一人として（私自身も含めて）、この数字を信用する人はいませんでした。

しかし、この頃になって、このいい加減な数字が、ひょっとするとここ二、三年で実現するかもしれない、というような気持ちになってきました。

もしそうなれば、すなわち、意識してマイクロコンピュータを使う人口が百万の大台に近づくようなことになれば、マイクロコンピュータはこれまで科学が生み落とした技術のなかで、人類にもっとも大きな影響を与えた技術の一つになることは間違いないと思います。

20世紀の最後を飾るにふさわしい、歴史的な出来事がいま、まさに起こったといえるでしょう。

そういったブームはまことに喜ばしいことですが、教育の現場で陣頭指揮をとっていますと、手放しで喜んでばかりはいられないことが沢山あります。

内地留学の制度を利用してマイクロコンピュータ技術の習得に来る他校の先生方や民間の技術者、それから研究室に配属される学生の指導を通じて、マイクロコンピュータが華やかに脚光をあびているわりには、基礎技術をしっかりとし身につけた人が少ないと痛感しております。

個々の例にふれると差しさわりがありますのでやめておきますが、私は、毎日本当に信じられないようなとんでもない出来事に直面させられて、目のまわるように忙しい、しかし一方で楽しいともいえる日々を送っております。

4 ま え が き

す。

初心者の幼稚な失敗を見て、笑ったり非難したりすることは簡単にできますが、さてそういった人々にマイクロコンピュータ技術を習得させるにはどうしたらよいかと考えると困惑することが沢山あります。

なかでも、一番困るのは、現在の時点でマイクロコンピュータのインターフェースにかんする初心者用の良いテキストがないということです。

たしかに、LSIメーカーの出している資料のなかには、良いものが沢山あります。しかし、こういった資料はあくまでも、一応でき上がった技術者を対象にしています。読者として、マイクロコンピュータを設計できるプロを想定しています。これは当然のことですが、この世には、ディジタル回路を見たこともない人が沢山いるのであって、こういった人にマイクロプロセッサの英文マニュアルを与えてみても、とうてい理解できるはずはないのです。この空白を埋めるために、本書を書く決意をしました。

インターフェース技術はマイクロコンピュータ設計の頂点にある技術です。インターフェース回路を設計するに当たっては、マイクロプロセッサを自由に使いこなす技術とともに、コンピュータをシステムとしてとらえる力も必要になります。

そういった関連技術をすべてこの小冊子のなかにもり込むのは、どう考えても無理です。そんなことをしたら、インターフェース回路について述べる部分が無くなってしまいます。

そこで、今回は思い切って関連部分をカットすることにしました。ソフトウェアの面とCPUまわり、たとえばメモリとかフロッピーディスクの制御回路などは他書にまかせることにしました。

そして、本書はマイクロコンピュータの応用回路の製作事例に専心させました。この方針が良いのか悪いのか読者の批判を待つことにしますが、いずれにしても、ここで述べた回路は私の研究室で十分に使い込んだものばかり

です。インターフェース回路を製作する際の参考になることを祈ります。

昭和52年10月

名古屋市にて 大川善邦

目 次

第 1 章 インターフェース回路を作るための基礎

1 . 1	インターフェースとは何か	9
1 . 2	データバス	10
1 . 3	アドレスバス	14
1 . 4	データバスからの情報の入力	16
1 . 5	双方向性のデータバス	27
1 . 6	バス制御線	31

第 2 章 マイクロプロセッサのバスの実例

2 . 1	はじめに	35
2 . 2	8080A のバス	36
2 . 3	8085 のバス	59
2 . 4	バスを作るときの注意	66

第 3 章 入出力タイプライタのインターフェース

3 . 1	はじめに	73
3 . 2	印刷装置のインターフェース回路	75
3 . 3	紙テープパンチ装置のインターフェース回路	86

8 目 次

3.4	紙テープ読み装置のインターフェース回路	89
3.5	キーボード読み装置のインターフェース回路	92
3.6	低速バスのインターフェース回路	95
3.7	データライタのインターフェース回路と基本サブルーチン	104

第4章 高速入出力装置と割込み処理

4.1	はじめに	109
4.2	割込み処理	110
4.3	時計回路と割込み処理プログラム	118
4.4	ドット型高速プリンタのインターフェース回路	125
4.5	手動式紙テープリーダーのインターフェース回路	136

第5章 機械制御のインターフェース回路とDMA

5.1	はじめに	153
5.2	電気自動車の制御	153
5.3	直流モータのインターフェース回路	157
5.4	ロータリーエンコーダのインターフェース回路	175
5.5	DMAによる情報の転送	185
5.6	ノイズと誤動作	200
あとがき		203

付 録

● 8085	211
● SCHOTTKY BIPOLAR 8228	221

第 1 章

インターフェース回路を作るための基礎

1.1 インターフェースとは何か

いま仮りに、どこかの都市の郊外に、新しく住宅地を建設するものとします。最初にどんなことを考えますか。個々の住宅やビルをどんなふう建てて、その内装をどうするかということも大切ですが、それにもまして、道路や輸送機関をどこに、どのように造っていくかが重要な問題になります。

コンピュータの場合もそうであって、中央処理装置 (Central Processing Unit ; CPU) のなかの演算回路やレジスタの配置を決めることも大切ですが、それにもまして、プロセッサが外の世界と情報を交換する道を定めるのが重要です。

この分野の技術を総称してインターフェース技術といいます。

インターフェース回路はコンピュータの頭脳となる CPU と手足となる入出力機械の接触点になりますから、インターフェース回路設計者は電氣的な知識と同時に機械的な知識を身につける必要があります、コンピュータのハードウェアとソフトウェアの両面の知識をもたなければなりません。

そのうえ、幅の広い教養と柔軟な頭脳も必要であり、何ものにもくじけることのない不屈の闘志が要求されるでしょう。

これまで、インターフェース技術はコンピュータ・メーカーのドル箱でした。アマチュアには近づくことのできない禁断の地だ、と喧伝されていまし

10 第1章 インターフェース回路を作るための基礎

た。

しかし、マイクロコンピュータの出現はインターフェース技術をそういったひとにぎりの老かいな職人的技術者の手からもぎとって、誰でも簡単にアプローチできるもっともポピュラーな技術の一つにしました。すべてが、明るい太陽のもとにさらけだされたのです。

こんなおもしろい出来事は、技術の歴史の上でも、そうざらにあることはありません。われわれも、この際、積極的にマイクロコンピュータのインターフェース回路作りに参加しなければなりません。

1.2 データバス

マイクロコンピュータのインターフェース回路を設計するに当たって、まず最初に、プロセッサからどのような形で情報が入出力されるか、その形式を理解しなければなりません。この形式は個々のマイクロプロセッサによって、多少違うことがあります。が、本質的なところは同じです。基礎をしっかりと身につければ、どんなマイクロプロセッサのインターフェース回路の設計も簡単にできるようになります。

まず最初に、マイクロコンピュータの情報の幹線となるデータバス (data bus) について述べます。

一般に、一つのマイクロプロセッサが、一つの基本的時間のなかで処理する情報の量はあらかじめ固定されています。4ビットのマイクロプロセッサは4ビットの情報を一時に処理し、8ビットのプロセッサは8ビットの情報を処理します。

ごくわずかの例外を除いて、マイクロプロセッサのデータバスは、この一時に処理できる情報と同じ本数の信号線をもちます。4ビットのプロセッサは4本のデータバスを、8ビットは8本、12ビットは12本、16ビットは16本

のデータバスをもちます。

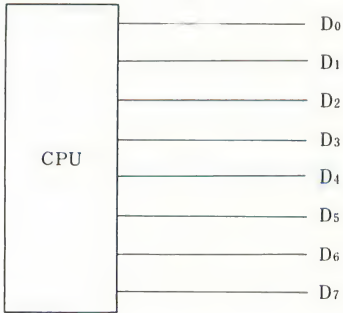


図1.1 8ビットのマイクロプロセッサのデータバス

図1.1は、8ビットのマイクロプロセッサのデータバスを概念的に書いたものです。8本のデータバスの信号線を区別するために、これらに D_0 , D_1 , D_2 , …… D_7 という記号をつけます。

計算機の世界では、0と1のいずれかの状態を取り得る信号を情報の単位に取り1ビット (bit) と呼びます。1ビットの信号が8本ままとすると、これを1バイトと

いいます。

1バイトのデータの各ビットは、各々独立に8個の情報を表現することもあります。多くの場合、数のように、互いに関連して一つの情報を表わします。このとき、8ビットの信号線の順序関係が重要になるので、もっとも最上位のビットをMSB (the Most Significant Bit) といい、最下位のビットをLSB (the Least Significant Bit) といって区別します。ここでは D_7 がMSB、 D_0 がLSBと約束します。すなわち、プロセッサ内部のレジスタとデータバスの対応は図1.2のようになると仮定するわけです。こう約束すると、 D_0 が1でその他が0の情報は数の1となり、 D_7 が1でそ

MSB

LSB

の他が0の数は+128となります。

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
-------	-------	-------	-------	-------	-------	-------	-------

最上位

最下位

データバスの各ビットをとくに区別する必要のない時には、

図1.2 データバスとレジスタのビット位置の対応

図1.1のように8本の信号線を

書かないで、図1.3のように簡略化して書くことがあります。そのほうが図面が見やすいことがあります。

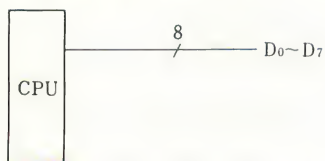


図 1.3 データバスを束ねて書いた場合

さて、いまプロセッサが外部に情報を送り出す場合を考えます。たとえば、外界に SN 7474 のような D タイプのフリップフロップが 8 個あってこれにプロセッサのなかのアキュムレータ (Accumulator または A レジスタ) の内容を転送したいとします。

この場合、プロセッサは A レジスタの内容をデータバスに出し、それからしばらくして書込みクロックとなるパルスを出し、これによってデータバスの情報を D フリップフロップに引き取らせ、その後の適当な時刻にデータバスの内容を引っ込める (次の内容に変更する) ことになります。

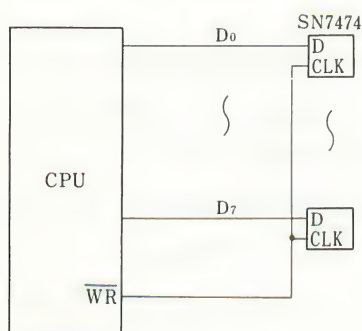


図 1.4 情報のアウトプットの操作

この状況を図 1.4 に示しました。ここでデータバスが 8 本ならば、D フリップフロップは 8 個までつけられますが、書込みのタイミングを与えるパルス \overline{WR} は 1 本でよいでしょう。

SN 7474 はクロック (CLK) が 0 から 1 に遷移するとき、入力端子 D の状態を取り込みますが、このときデータはクロックの立上りの前後で安定していなければなら

りません。すなわち、クロックが立ち上がる前にすくなくとも t_s 時間はデータが安定して固定される必要があります。同様に、クロックが立ち上がった後も最小限 t_h 時間データを保持しておかなければなりません。これを図 1.5 のように書きます。この図で、 t_s のことをデータのセットアップ時間 (data set up time) といい、 t_h をデータのホールド時間 (data hold time) といいます。

もし、この条件が破られると、すなわちクロックの立上りの前後でデータ

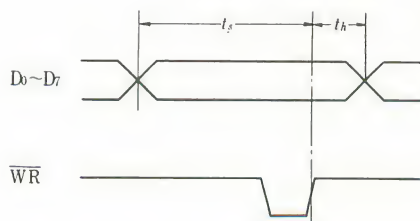


図 1.5 セットアップ時間とホールド時間

を変化させると、一度セットされたフリップフロップがクロックも入らないのに反転したり、不安定な動作が起こります。ちなみに、テキサスインスツルメント社の資料によると SN7474 の t_s は最小で 20 nsec, t_h は最小で 5 nsec

と定められています。

マイクロコンピュータの場合、フリップフロップにデータをセットするよりも、メモリにデータを書き込むときのほうがクリティカルな状態になるので、実際問題としては、こちらの設計に気がつかいます。

図 1.4 のクロック信号は \overline{WR} と書きましたが、このように英字の上に (バー) を乗せた記号のことを負極性 (active low) の信号といいます。この信号は常時 1 の状態になっていて、いざ動作をしたいというときに 0 の状



図 1.6 正・負極性のパルスの比較

態になります。図 1.6 に正極性 (active high) のパルスと負極性のパルスの違いを図示しました。

データバスの書込みのタイミングを示すパルス \overline{WR} のように、長い時間のなかで、ごくわずかの時間だけ有意な信号を送り出す信号線は、普通、負極性の状態にしておきます。

その理由は、第 1 に、TTL 回路の信号の範囲が 0 よりも 1 のほうに余裕があるためで、もし外界から混入してくるノイズが同じ大きさで、0 にも 1 にも同じように加法的に加えられるとすれば、長い時間を状態の 1 にして、

ごく短い時間を0にしたほうが系の耐ノイズ性が増加します。統計的に考えればその妥当性が理解できるでしょう。

第2に、TTL回路の特性によって、バスに0を出力するにはドライバ素子がバスから電流を吸い込まなければなりません、1の場合はそういった必要がないのでむだな電力の消費をしないですみます。

信号線 \overline{WR} のように、データバスの上の情報の転送を助ける役目をこなう信号線をとくにバス制御線といいます。

1.3 アドレスバス

マイクロプロセッサが情報を送り出す対象は、一般に、一つではなく複数あります。たとえば、メモリが4 Kバイトあるというときには、 $4 \times 1024 = 4096$ バイトのフリップフロップ（あるいはそれと同等の記憶素子）があり、プロセッサはそれらのどのセルに現在情報を送り出しているかを指定しなければなりません。

何千とか何万とかの対象を区別するには、符号化（encoding）という操作を採用します。すなわち何本かの信号線を用意し、これをたとえば n 本とすると、この信号線に2進数の

$$0 \sim 2^n - 1$$

の数を置き、その数に対応して対象を区別します。

この数を番地（アドレス）またはポート番号などといい、そのアドレスを乗せる n 本の束線をアドレスバス（address bus）といいます。

マイクロプロセッサは16本のアドレスバスをもつのが標準のようです（ $n = 16$ ）。このとき

$$2^{16} = 65,536$$

ですから、プロセッサは65,536個の対象を区別できることになります。

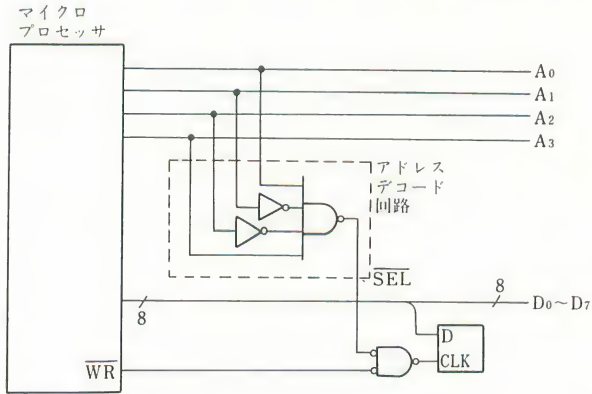


図 1.7 アドレスのデコード回路

アドレスバスの信号線に A_0, A_1, \dots, A_{15} の記号をつけます。MSB は A_{15} , LSB は A_0 とするのが普通です。たとえば、9 番地を指定するには、 A_0 と A_3 を 1 にして、その他を全部 0 にします。

いま、図 1.4 のフリップフロップにたいして、アドレスの 9 番を割り当てたとします。簡単のために $n = 4$ としたときの様子を図 1.7 に示しました。

この図で、 A_3 と A_0 はそのまま、 A_1 と A_2 は NOT 回路を通して 4 NAND 回路に入れます。これをアドレスデコード回路といいます。4 NAND の出力端に $\overline{\text{SEL}}$ という記号をつけましたが、これは論理式で

$$\overline{\text{SEL}} = A_3 \cdot \overline{A_2} \cdot \overline{A_1} \cdot A_0$$

と書けます。 A_3, A_2, A_1, A_0 が 1001 になったとき、そのときにかぎって $\overline{\text{SEL}}$ は 0 になります。そうでないとき、 $\overline{\text{SEL}}$ は 1 になっています。

そこで、 $\overline{\text{SEL}}$ と $\overline{\text{WR}}$ を図のように OR 回路に入れると

$$\text{CLK} = \overline{\text{SEL}} + \overline{\text{WR}} = \overline{\text{SEL} \cdot \text{WR}}$$

となって、アドレスバスに 1001 が出て、かつ WR のパルスが出たときに、D フリップフロップにクロックが入るということになります。ここで OR 回路を負極性の信号の NAND 回路と書いたのはブール代数の二重否定の定理と

16 第1章 インターフェース回路を作るための基礎

ド・モルガンの定理を使ったためで、

$$X + Y = \overline{\overline{X + Y}} = \overline{\overline{X} \cdot \overline{Y}}$$

となるからです。この意味がわからないときは、論理数学の参考書を勉強して下さい。

マイクロプロセッサが外界に情報を転送する場合の手順をまとめると、次のようになります。

- (1) マイクロプロセッサは情報の転送先の番号をアドレスバスに出力する。
- (2) 転送する情報をデータバスに出力する。
- (3) 適当な時刻に書込みのパルスを出力する。

細かい時間タイミングの問題は次章で具体的にマイクロプロセッサの性質を調べるときに考えることにします。ここでは、手続きのしくみを理解すればよいことにしておきましょう。

1.4 データバスからの情報の入力

次に、外界の情報をマイクロプロセッサの内部に取り込む場合について考えます。

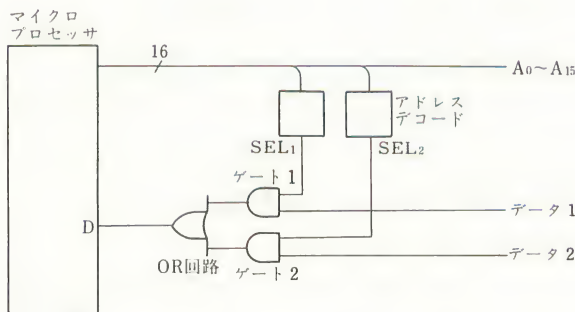


図 1.8 外界から情報の取り込み

原理的にいえば、外部の情報がすべてプロセッサのなかに入らなければならないので、図 1.8 に示したように、外部の情報がアドレスバスのデコード信号でゲートされ、その結果が OR をとられてプロセッサの端子に入ることになります。

図 1.8 において、もしアドレスバスにデータ 1 の側のアドレスが出ると、アドレスデコード回路の出力は

$$\text{SEL}_1 = 1, \text{SEL}_2 = 0$$

となります。したがって、この場合、データ 2 の 1 か 0 かにかかわらず、ゲート 2 の AND 回路の出力は 0 になります。一方で、ゲート 1 は SEL_1 が 1 だから、データ 1 の 1 か 0 かがそのまま OR 回路に入り、結局、これがマイクロプロセッサに取り込まれていくことになります。もし、アドレスバスにデータ 2 の側の番号が出れば、データ 2 の情報がプロセッサに取り込まれます。

原理的にいえば図 1.8 でよいのですが、現実の問題としては、入力が多い OR 回路をどう構成するかがポイントになります。

最初に、オープンコレクタゲートを使用した OR 回路について考えてみますが、そのまえにまず、標準 TTL ゲートの動作原理について述べます。

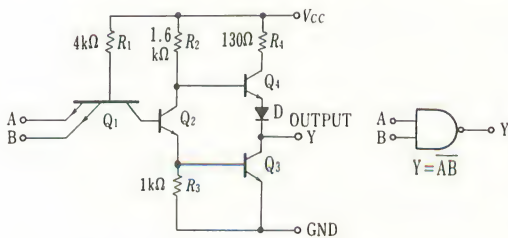


図 1.9 (a) スタンダード TTL 回路

図 1.9(a)は TI 社のマニュアルからとった標準 TTL 素子の等価回路です。一番左側の Q_1 のトランジスタに A と B の 2 本のエミッタがついていて、このエミッタの各ピンに入力信号が与え

られるようになっています。 Q_1 をマルチエミッタのトランジスタといいます。

このマルチエミッタのどれか一つが、いま、接地されたとしてみます。そ

18 第1章 インターフェース回路を作るための基礎

うすると、 Q_1 のベースは $4\text{ k}\Omega$ の抵抗を通じて V_{cc} (普通5ボルトにする)に吊り上げられていますから、ここから電流が流れ込み、ベースからエミッタに抜けていきます。そうすると、 Q_1 は導通の状態(オンの状態ともいう)になり、コレクタからエミッタに向けて電流が流れます。

ところで、 Q_1 のコレクタは Q_2 のベースに入っていますから、 Q_1 がオンになるということは、 Q_2 のベースからエミッタに向けて電流を流さないということになります。すなわち、 Q_1 がオンになれば、 Q_2 はオフ(しゃ断状態)になります。

Q_2 がオフになると、 Q_3 のベースは $1\text{ k}\Omega$ の抵抗を通して接地されますから、 Q_3 はオフとなります。一方で、 Q_4 は $1.6\text{ k}\Omega$ の抵抗を通して V_{cc} から電流がベースに、そしてエミッタに流れていきますから、 Q_4 はオンとなります。

要約すると、入力マルチエミッタのすくなくとも一つの入力線が接地されると、出力トランジスタ Q_3 と Q_4 のうち、 Q_4 がオンとなり Q_3 がオフとなるということになります。このとき、出力の電位は高くなり、電流はこの回路から次段の回路に流れ出ようとします。これをソース電流(source current)といいます。

次に、全部の入力ピンを高い電位(たとえば V_{cc})につないだとします。このとき、 Q_1 のトランジスタはオフの状態となりますが、電流はベースとコレクタのPN接合を通じて流れます。これをトランジスタが逆能動領域(inverse active region)にあるといいます。

このとき、 Q_2 のトランジスタはベースからエミッタに電流が流れ、オンの状態になり、コレクタからエミッタに電流が流れます。

そうすると、 Q_4 のトランジスタのベース電位は $1.6\text{ k}\Omega$ の抵抗に電流が流れただけドロップしますから、オフの状態になります。 Q_3 のトランジスタはその逆で、 $1\text{ k}\Omega$ の抵抗に電流が流れただけベース電位が上がりますので、オンの状態になります。

要約すれば、入力ピンの全部を高い電位にすると、 Q_4 がオフで Q_3 がオンになります。出力端子の電位は下がり、次段の回路から電流を吸い込む状態になります。これをシンク電流 (sink current) といいます。

出力段に Q_3 と Q_4 の2つのトランジスタがつみ重ねられているところがTTL回路の特徴であって、これをトーテムポール (totem-pole) とかアクティブプルアップ (active pull up) といいます。

TI社の規格を見ると、入出力信号の電位は次のように定められています。

V_{IL} ; 入力端において論理0となる電圧レベル、0.8V以下でなければならない。

V_{IH} ; 入力端において論理1となる電圧レベル、2V以上でなければならない。

以上の2条件が守られたときに、TTL回路はその電圧を信号として正しく認識し、次のような出力電圧を出します。

V_{OL} ; 出力端において論理0となる電圧レベル、0.4V以下になることが保証されている。

V_{OH} ; 出力端において論理1となる電圧レベル、2.4V以上になることが保証されている。

出力端の電圧レベルよりも、入力端の電圧レベルのほうが幅が広いということは、信号が伝送される途中で雑音が入っても、その大きさがある範囲以下ならば、信号は正しく処理されることを意味しています。

$$2.8 - 2.4 = 0.4$$

$$0.8 - 0.4 = 0.4$$

となりますから、通信路で混入してくる雑音が0.4V以下ならば、信号は正しく伝送されることになります。

この電圧差を保証された雑音余裕 (guaranteed noise margin) といいます。雑音余裕以上の電圧が混入するとゲートの動作はどのようになるか保

20 第1章 インターフェース回路を作るための基礎

証されません。だから、そのようなことのないように、回路を設計しなければなりません。

以上は電圧について考えたわけですが、電子回路において、電圧は回路を流れる電流の結果として生まれてくるものです。だから電圧に定められた条件を守るためには、一つの出力端にどのような入力端を接続したらよいかという問題になります。

一つの出力端が1を出力しているとき、次の2種類の電流が関係してきます。

I_{load} ; 最悪の場合でも、駆動側で、とにかくこれだけの電流は流すことができる電流、普通のTTLゲートでは $400\mu A$

I_{in} ; 最悪の場合でも、駆動される側で、これ以上の電流は吸収しないという電流の最大値、普通のTTLゲートでは入力端1本につき $40\mu A$

ここで符号は、いま考えている回路から出ていく電流を負で表わし、その回路に入ってくる電流を正で表わします。このように作られているTTL回路では

$$400 \div 40 = 10$$

となりますから、一つの出力端にたいして、状態1のとき、最大10個のゲートをつけることができることになります。

駆動側が出力0を出しているときのパラメータは次のようになります。

I_{sink} ; 最悪の場合でも、駆動側でとにかくこれだけの電流は吸収することができるという電流の最大値、普通のTTLゲートでは16mA

I_{L} ; 最悪の場合でも、駆動される側でこれ以上の電流は出さないという

電流の値、普通は1.6mA

したがって、次のように計算すると

$$16 \div 1.6 = 10$$

となって、こちらも一つの出力端に10

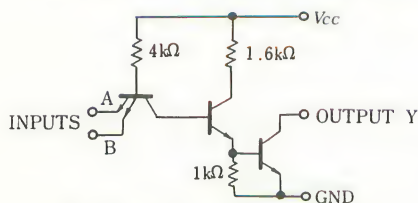


図 1.9 (b) オープンコレクタゲート回路

個の入力端子を接続できることになります。

以上はスタンダード TTL 回路の動作原理を説明したのですが、このトータムポールトランジスタの下だけを残して上を取りはらい、コレクタを直接出力に出したものがオープンコレクタゲートです (図1.9(b))。

出力トランジスタがオンのとき、外部から電流を吸い込みますが、オフのときには電流をほとんど取りません。近似的に、オフのトランジスタは出力ラインから切り離されていると考えてもよいでしょう。

入力から出力への信号の伝送は、オープンコレクタもトータムポールも同じですから、オープンコレクタゲートの入力の一つをロウに落とすと、出力トランジスタがオフになります。

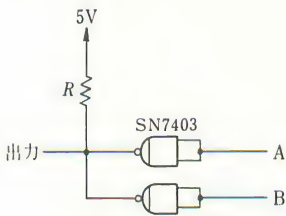


図1.10 ワイヤード OR 回路

いま、オープンコレクタゲート SN7403 を図1.10のように出力端を結び抵抗 R で電源に吊り上げたとします。このとき、両方の出力トランジスタがともにオフならば、抵抗 R を通して電流はほとんど流れませんから、出力は高い電位、すなわち状態の 1

となります。

AかBのすくなくとも一方のトランジスタがオンの状態になると、電流は5ボルトから R を通ってそのオンのトランジスタに流れ込み GND にぬけていきます。電位は抵抗 R によって降下し、出力は低い電位、すなわち状態の 0 になります。

どれかのトランジスタがオンになれば出力が 0 になるというわけですから、トランジスタがオンになる現象と出力が 0 になる現象に着目すれば、これが OR 回路になっていることがわかります。

もし、出力トランジスタがオフになることと出力が 1 になることに着目すると、図1.10の回路は、すべてのトランジスタがオフのとき出力が 1 になる

22 第1章 インターフェース回路を作るための基礎

回路だから、これは AND 回路であるといえます。

物理的現象は1つに決まっていますが、信号の命名法によって、AND回路にも OR 回路にもなります。もし信号の0に着目することに慣れていないときは図1.11の回路のように書くと、AかBのいずれか一方を1にする

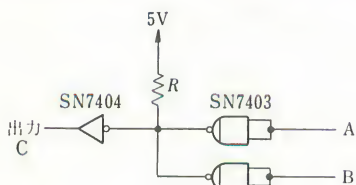


図 1.11 OR回路 $C = A + B$

と出力が1になる回路となって考えやすくなります。

さてそこで、オープンコレクタゲートを使ってマイクロプロセッサの入力バスを作ると、たとえば図1.12のようになります。

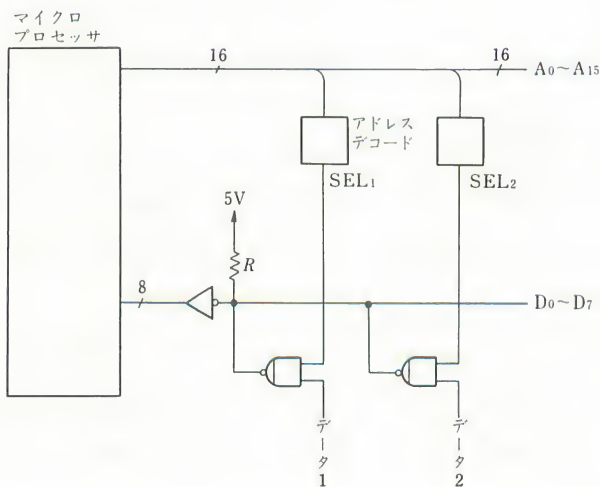


図1.12 オープンコレクタゲートによる入力バス

アドレスバスからデコード回路を通して選択信号 SEL_1 と SEL_2 を作ります。この信号はアドレスバスにある特定の数が置かれたとき、そのときだけ1になり、その他の場合はゼロになります。

したがって、選択されないオープンコレクタゲートは入力的一方に0が入

るので、出力トランジスタは必ずオフになります。選択されたゲートは、選択信号が1となるので、データが1のときに出力トランジスタがオン、データが0のときに出力トランジスタはオフとなります。出力トランジスタがオンのとき、電流を吸い込んで出力電位はロウになりますが、これがNOT回路によって反転されますから、結果として、マイクロプロセッサにはデータがそのまま取り込まれることになります。

オープンコレクタゲートを使った入力バスは、今から10年ぐらい前までは盛んに使用されていたのですが、1970年頃に、ナショナルセミコンダクタ社が3状態 (TRI STATE) 素子を発表するにおよんで、バス構造は急速にこの3状態素子を採用するようになりました。

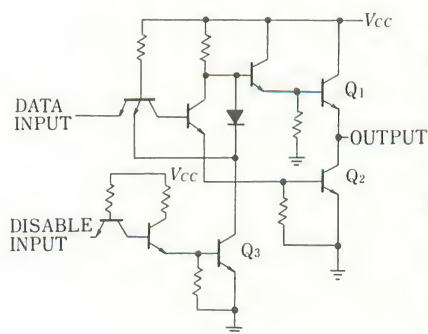


図 1.13 3状態素子ゲート回路

3状態素子の動作原理図をNS社の資料から再掲すると図1.13のようになります。この図を見ると、出力のトータムポールトランジスタ Q_1 と Q_2 は標準TTL回路と同じになっていますが、トータムポールの上のトランジスタがダーリントントランジスタになっていることと特殊な禁止入力がついているところが違います。

いま、禁止入力をハイにしたとします。トランジスタ Q_3 はオンになりますから電流を吸い込みますが、これによってマルチエミッタ入力の一つはゼロに落ち、かつダーリントントランジスタのベースもゼロに落ちます。

入力のマルチエミッタの一つにゼロが入ると、TTLゲート回路の特性によって、トランジスタ Q_2 はオフになりますが、同時に、ダーリントントランジスタのベースがゼロに落ちているので Q_1 がカットオフになります。すなわち、トータムポールトランジスタの上下が共にオフになります。もし、禁止

24 第1章 インターフェース回路を作るための基礎

入力をロウに落とすと、 Q_3 がカットオフになって、普通のTTL回路と同じになります。

これまでのTTL回路は

(1) Q_1 オンで Q_2 オフのとき出力1

(2) Q_1 オフで Q_2 オンのとき出力0

という2つの状態しかなかったのに、新しく

(3) Q_1 オフで Q_2 もオフとなって、出力は0とも1とも定義されない電圧となる。

という第3の状態を導入することに成功しました。これによって、NS社は第3の状態という意味でTRI STATEという名前をつけたのです。

上下のトランジスタが共にカットオフになると、ごくわずかの電流は出入りしますが、この電流はかなり小さい値なので、近似的には、そのゲートがバスラインから切り離されたと見てもさしつかえないようになります。

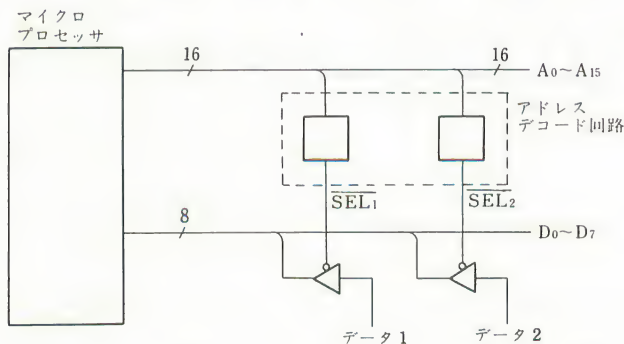


図 1.14 3 状態素子を使った入力バス

そこで、図1.14のように配線すれば期待していたデータの入力バスができることとなります。

アドレスデコード回路は、選択されたアドレスだけが0に落ちるように作らなければなりません。そうしておいて、いま仮りに、データ1のアドレス

がアドレスバスに乗ったとすると、 $\overline{\text{SEL}}_i$ が 0 となり、その他の選択信号はすべて 1 になります。

このとき、前に説明した 3 状態素子の特性によって、データ 1 以外のゲートは第 3 の状態、すなわち上下のトータムポールトランジスタが共にカットオフになる状態、近似的にバスから切り離されたような状態になります。データ 1 の禁止入力 is zero ですから、こちらは普通の TTL ゲートとして作用し、このゲートがバスをドライブすることになります。

こうして、複数のデータの切り替えが可能になります。これをデータのマルチプレックス (multiplex) ともいいます。

さて、最後に 3 状態素子のトータムポールトランジスタの上側がダーリントントランジスタになっていた理由について述べます。

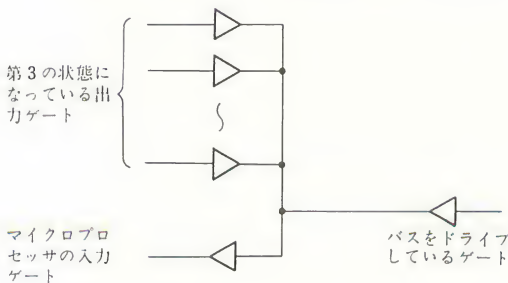


図 1.15 入力ゲートの状態

いま、一つのゲートがバスをドライブしている状態を見やすく書くと図 1.15 のようになります。ドライバの負荷となるのは

- (1) マイクロプロセッサの入力端
- (2) 第 3 の状態になって

いるゲートの出力端

の 2 種類です。数値的な目安をつけるために、ゲートはすべて標準 TTL ゲートと仮定し、第 3 の状態になったゲートの出力端の負荷電流は $40\ \mu\text{A}$ と仮定して計算をしてみます。

まず、ドライブ側のゲートがバスに信号の 0 を出力しているときですが、このとき TTL ゲートの特性によって、ドライバのトランジスタは $16\ \text{mA}$ まで電流を吸い込むことができます。プロセッサの入力ゲートは、いま、TTL

26 第1章 インターフェース回路を作るための基礎

ゲートと仮定していますから、これを状態の0に維持するために、ここから1.6 mAの電流を吸収しなければなりません。

第3状態のゲートは出力トランジスタが上下ともカットオフになっていますから、仮定によって、上のトランジスタから40 μ Aの電流がもれ出てきます。第3状態になっているゲートの数を x とすると

$$1.6 + 0.04 \times x \leq 16$$

という代数方程式が成立すればよいことになります。これを解くと

$$x = 360$$

となって、結局、1つのバス線にぶら下げることのできるゲートの数は自分自身も含めて361個ということになります。

同じようにして、ドライバが1を出力しているときのことを考えますと、受信のゲートが40 μ A、第3状態になっているゲートの下のトランジスタが各々40 μ Aの電流をとりますから

$$0.04 + 0.04 \times x \leq 0.4$$

という代数方程式を解くことになり、これを解くと

$$x = 9$$

となります。自分自身も含めて10個のゲートしかつけられない、ということになります。ドライバの出力は0にも1にもなりますから、361と10の小さいほう、すなわち10個が解になります。

これではあまりにもアンバランスなので、3状態素子のゲートではトータムポールの上のトランジスタをダーリントン接続にして出力電流を強化します。

こうして上のトランジスタを強化すると、今度は、下のトランジスタがクリティカルな状態になるので、設計はドライバが0を出力しているときの吸い込み電流の値によって決まることになります。

以上の計算はあくまでも電流の直流的な収支だけに目をつけたものです。

バスの上を速い信号が送られてくるときは容量性の負荷による信号の遅れが問題になりますから、注意しなければなりません。

1.5 双方向性のデータバス

これまで、説明を簡単にするために、バスのうえの情報の入力と出力に別別の信号線で行なわれるように説明してきました。ところが、もしマイクロプロセッサの情報の入力と出力がまったく同じ時刻に発生するのでなければ、一つのデータバスを使って、ある時刻には情報の入力、そして別の時刻には情報の出力というように使い分けることができます。

ちょうど、汽車の線路を複線にしないで、単線にしておいて、ある時刻には上りの列車を運行し、別の時には下りの列車を走らせるようなものです。汽車の場合は列車が終点につくのに相当の時間がかかりますが、電気的な信号の場合はごくわずかの時間しかバスを占有しませんから、このようにしたほうがプロセッサから出る信号線の数を少なくすることができます。

一つの路線を、あるときは入力情報が占有し、他の時刻には出力情報が占有することを時分割あるいはタイムシェアなどといいます。一つのバスから、ある時刻にアドレス情報が出力され、次の時刻にデータが入力されるようになるとき、アドレス情報とデータが一つのバスを共有していることになりませんが、このときも時分割、あるいはタイムマルチプレックスといいます。

さて、一つのデータバスにおいて、情報の入出力を時刻を分けて実行するとき、まず、図1.16のように、一つのバス信号線に複数個のドライバ素子とレシーバ素子を接続します。

レシーバ素子がバスから信号を取り込む操作を禁止することはできないので、この素子はバスを通過する信号に従って、バスから電流を取ったり、バスに電流を出したりします。ということは、すべてのレシーバは、常時、バス

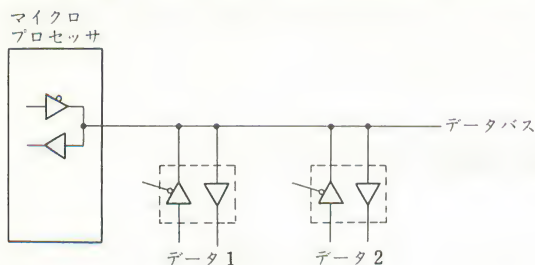


図 1.16 双方向性のバス構成

まず、マイクロプロセッサが信号を外界に送り出す場合ですが、これは既に説明したように、アドレスバスにアドレス情報を乗せ、これで外界の複数の装置のなかの一つを選択し、次に、マイクロプロセッサのドライバ素子がバスを駆動し、データをデータバスに乗せ、そして最後に \overline{WR} という書込み信号を出し情報のセッティングをします。

このとき、注意しなければならないのは、バスにぶら下がっているドライバ素子のなかで、プロセッサ以外のものはすべて第3の状態になっていなければならないということです。その理由を理解するために図1.17を見て下さい。

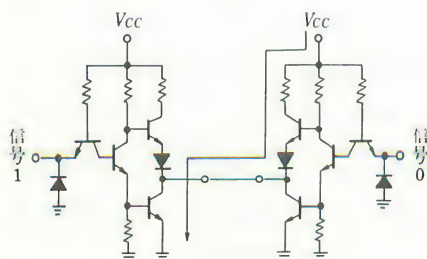


図1.17 出力を接続したTTLゲート

かをオン (active) にします。もし、このオンになっているトランジスタが左右両方とも上とか下というように一致していれば問題はないのですが、逆になった場合は、電源がショート (短絡) された状態になりますから、オン

線の負荷になるということとを意味しています。これは後に、マイクロコンピュータのバスを設計するときに必要になりますから記憶しておいて下さい。

これはバスに接続されている2つのドライバ素子の出力端を示したのですが、もし、両方のドライバのイネーブル入力と共にハイになっていると、2つのドライバはバスを駆動しようとして、トータムポルトランジスタのどちら

とオンのトランジスタのなかを大きな電流が流れ、発熱量が上がリ、トランジスタがダメージを受けます。図1.17は右側のゲートに0が入り、左側のゲートに1が入力されたときの電流の短絡の様子を示しています。

こういった理由で、プロセッサがバスをドライブするときには、データバスにぶら下がっている外界のドライバ出力は禁止されます。

これを逆にいうと、プロセッサが外界から情報を受けるとき、そのときだけ外部のドライバはバスを駆動することを許される、ということになります。すなわち、マイクロプロセッサのデータバスには接続されているドライバはほとんどの場合、出力禁止の第3状態になっていて、ある特定の信号（ここで仮りに \overline{RD} としておきます）がロウに落ちたときだけ、活躍を許される、ということになります。

勿論、プロセッサ以外のバスドライバのどの2つも同時にバスを駆動することはできませんから、図1.18のように、アドレスバスの情報と \overline{RD} の信号をデコードして、各ドライバのゲートを開けるようにします。

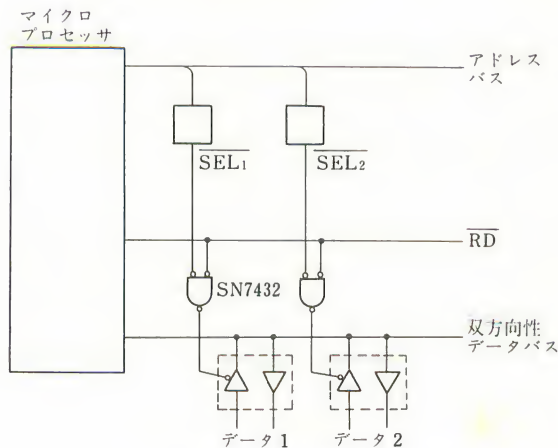


図 1.18 双方向性データバスの構成

30 第1章 インターフェース回路を作るための基礎

このように設計すれば、アドレスに一つの機器のアドレスが出力され、かつ \overline{RD} の信号が送られてきたときに、その機器のドライバがバスを駆動することになります。図のなかで使っているゲートはド・モルガンの定理と二重否定の定理によって

$$\overline{\overline{X} \cdot \overline{Y}} = \overline{\overline{X}} + \overline{\overline{Y}} = X + Y$$

となりますから、これは OR 回路（たとえば SN7432）となります。

以上の検討の結果をまとめると、マイクロプロセッサが外界と情報を交換するための基本として、次のような信号線が必要だということになります。

(1) アドレスバス（出力）

メモリの番地あるいは装置の番号（ポート番号ともいう）などを指定する情報が送り出される。マイクロプロセッサが情報を交換する相手を指定する。

(2) 双方向性のデータバス（入力または出力）

プロセッサがメモリや装置に送り出す情報、またはメモリや装置から受け取る情報が乗る。8ビットのマイクロプロセッサの場合、普通8本ある。

(3) バス制御信号（出力）

これまでの説明で、 \overline{WR} と \overline{RD} と呼んでいた信号線のことで、マイクロプロセッサが外部に情報を送り出すときの書込みパルスとして使う \overline{WR} と外部から情報を受けるときのゲートを開く \overline{RD} の2つがある。実際には、この他のバス制御信号があるが、それらについては次節で検討する。

ここで、入力と出力というのはマイクロプロセッサから見たときの情報の転送の形をいうのであって、見方を逆にすれば、入力と出力が逆になることに注意しなければなりません。

1.6 バス制御線

以上でインターフェース回路の幹線となるアドレスバスとデータバスの働きのしくみの説明は終わりますが、バス制御線についてももう少しつけ加えておきたいと思います。

バス制御線の例として、すでに $\overline{\text{RD}}$ と $\overline{\text{WR}}$ ができました。こういったバス制御線は、ちょうど道路ぎわに立っている交通信号のようなものであって、交通の流れをスムーズにし衝突事故などが起きないように、情報の流れをコントロールする役目を果たしています。

$\overline{\text{RD}}$ と $\overline{\text{WR}}$ のバス制御線の他にいくつかの制御線がありますから、それらを順々に説明していきます。

一般に、マイクロコンピュータは並べられた命令を順々に実行していきます。これでとくに不都合はないのですが、とくに処理の能率をあげようとする、もう少し工夫する余地ができます。

いま例として、マイクロコンピュータが1秒ごとに外界の電圧を読みとって、記憶装置のなかにしまうものとします。

このときに、もし、マイクロコンピュータが1秒待ってデータを読み込み、それをメモリにしまって、そしてまた1秒まってデータを読み込み……というようにデータの読み込みに専心したとすればなんの問題もありません。

しかしよく考えてみると、データを読み込み、それをメモリにしまうという操作はおそらく20 μ 秒もあれば十分に実行できます。マイクロコンピュータは100万回のチャンスのなかで、たった20回だけしか有効に使っていないということになります。

これはちょっともったいない気がします。もし、たとえばお菓子屋さんが郵便局まで行かなければならなくなったとして、自分の店を閉めて郵便局に

行き、用を済ませて帰ってきて店を開ける。今度は、市役所まで用事ができて、また店を閉めて出て行ってしまったらどうでしょうか。能率が悪くてお客さんも困ってしまうでしょう。こういうときには、誰か手のあいてる人に用事をしてもらって、商店の営業は続けていくというのが常識です。

マイクロコンピュータの場合もそうであって、1秒ごとにデータをサンプルする例の場合であれば、プロセッサは何か他の処理をしていて、1秒ごとに外部から信号をもらい、そのときだけ仕事を中断してデータの取り込みを行ない、それが終わったら、また中断した仕事を継続するというようにします。これを割込み (interrupt) といいます。

マイクロコンピュータにたいして、割込みを可能にするために、バス制御線のなかに、普通、割込み要求線とその割込み要求がプロセッサによって受け付けられたことを出力する線が用意されます。

割込み要求があったときに、マイクロプロセッサがこれに答える過程はプロセッサの種類によってかなり違いますから、一般的に述べるのは困難です。ここではバス制御線のなかに割込みに関係する信号線があるということを指摘するだけにとどめておいて、次章で実際のマイクロプロセッサの性能について検討するときに、もっと詳しく述べることにします。

一般論として、いまいえることは、マイクロプロセッサが割込み要求を受け付けると、その次に実行すべきはずだった命令の番地をどこかにしまっておいて、他の番地に飛んで行って別の命令を実行し、それが終わったら、しまっておいた番地を復活して、中断した処理を再開するということです。

もし、割込み処理について知るのがこれが初めてだという人は、プログラミングの入門書を手に入れて、割込み処理プログラムの作り方のところを勉強する必要があると思います。

次に、マイクロコンピュータのバスをマイクロプロセッサの支配から取り上げて、一時的に他の装置がバスを占有する方法について述べます。

一般に、マイクロコンピュータのバスにはメモリとかいろいろな入出力装置が接続されています。このバスは、常時はマイクロプロセッサの支配下にあります。時には入出力装置が直接にメモリからデータを読み出したいとか、または他のコンピュータシステムが直接に、そのメモリや入出力装置にアクセスしたいということがあります。これを DMA (direct memory access) といいます。

この要求をみたすために、マイクロプロセッサには HOLD (ホールド) とか BUS REQUEST (バスリクエスト) という信号線が用意されています。

いま、仮りに外部の装置が一つのマイクロコンピュータシステムのメモリに直接アクセスしたいとします。このとき、その装置は最初にバスを占有したいという要求を、たとえばHOLDというバス制御線を通してプロセッサに出します。

マイクロプロセッサは、通常、命令を実行していますが、その適当な時刻に外部からのバス占有要求を検知すると、プロセッサが占有していたバスを切り離すために、アドレスバスとデータバス、それから種々のバス制御線のドライバ素子を第3状態にします。これは電氣的にいうと、プロセッサがバスを切り離したのと同じ効果をもたらします。そして、バスを切り離したということを外部に知らせるために、HOLD ACKNOWLEDGE (ホールドアックノリッジ) という信号線に信号を出します。

バス要求を出していた装置はプロセッサからバス切り離し完了信号を受け取ったら、今度は、こちらからアドレス、データ、バス制御信号を出して、必要な仕事をします。

仕事が終わったときに、ホールド要求を落とせば、マイクロプロセッサは中断したプログラムの実行を再開します。

このようにして、一つの幹線に、いくつかのプロセッサとか装置が相互乗り

34 第1章 インターフェース回路を作るための基礎

入れすることになります。バス占有要求を細かく出すと、いくつかのプロセッサが入り交ってメモリにアクセスすることになるので、これをサイクルスチーリング (cycle stealing) ということがあります。

このように、マイクロプロセッサのバス制御線には、バスの切り離しを要求する線と、その要求を受け付けたことを外部に知らせる信号線があります。

この他に、電源を入れたときに出力されるリセット線およびプロセッサの処理の進行を支配するクロック線がバスに送られることがあります。

以上の検討結果をまとめると、バス制御線として、次のような信号線があることがわかります。

- (1) データの書込みおよび読出しのタイミングを与える制御線
- (2) 割込み要求とそれが受け付けられたことを知らせる信号線
- (3) プロセッサにたいしてバスを切り離せという要求、および切り離し完了を知らせる信号線
- (4) 電源投入時などに出るリセット線
- (5) プロセッサの基本クロック線

インターフェース回路を設計することは、つきつめていえば、これらのバス信号線に支配された回路の設計ということになります。次章で、実際のマイクロプロセッサについて、信号発生の時間タイミングなどについて検討することになります。

第2章

マイクロプロセッサのバスの実例

2.1 はじめに

インターフェース回路はマイクロプロセッサと機器をつなぎ合わせるものですから、それを設計するに当たって、マイクロプロセッサのハードウェアとソフトウェアの構造と、それからそれに接続する機器を動かすしくみを理解しなければなりません。近くの書店に行って、インターフェースという名前のついている書物を二、三冊手にとって頁をめくってみました。いずれも本の半分以上が CPU の構造と命令語の解説についやされていました。

これは、一冊の書物で、できるかぎり完結したインターフェース回路の本を書こうと思うと、やむを得ないことなのですが、一方で読者の立場に立つと、インターフェース回路の本を買うごとに、プロセッサの解説を買い込んでしまうことになります。

そういった点を考慮し、あれやこれや考えた結果、今回は思い切ってマイクロプロセッサの解説は削除することになりました。ここではマイクロプロセッサとして、8080A、8085を取り上げますが、これらのプロセッサの詳しい構造については別の書物やメーカーのマニュアルにゆずることにします。

しかし、そうはいつでも、マイクロプロセッサについてまったく無知ではインターフェース回路を設計することはできません。この章で、これらマイクロプロセッサをバス構成という立場から、その構造を調べてみることにし

ます。マイクロプロセッサをバスから見るというのが、ここでとる基本的な思想です。

2.2 8080 A のバス

マイクロプロセッサの最初の例として、インテル8080 Aを取り上げます。このプロセッサは長所も短所ももっていますが、何をいうにしても、8080がマイクロコンピュータの歴史で一時期を画したものだということは否定できません。そういった意味で、ここで、8080 Aをインターフェース回路を設計するという立場から眺めることにします。

8080 Aにかぎらず、マイクロプロセッサのなかの処理は、すべて、一連の基本的な刻時パルス列に拍子をとられて進行します。この刻時パルス列をクロック (clock) といいます。

8080 Aは ϕ_1 と ϕ_2 という2種類の非重複2相クロック (non overlapping 2-phase clock) を必要としますが (図 2.1), このクロックは8224というICで発生することができますから、ここでその規格の細かいことにふれる必要はないと思います。

ただ一つ図 2.1 の t_{cy} はクロックの周期ですが、こ

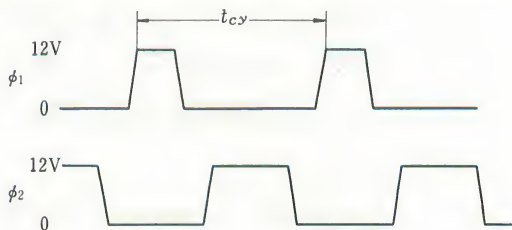


図 2.1 8080 A の非重複 2 相クロック

れは480nsec以上、2.0 μ sec以下と定められています。下限の480 nsecはLSIのなかの処理速度から、上限の2.0 μ secはレジスタ類にダイナミック回路を採用していることから制限されたものです。発振回路に18MHzのクリスタルを用いると、8224のなかで1/9に分周されるので、

$$18.00 \div 9 = 2 \text{ MHz}$$

となって、2MHzのクロック、もしくは $t_{cy}=500\text{nsec}$ でプロセッサの処理が進行するようになります。

インターフェース回路の設計からいうと、8224は ϕ_2 をTTLレベルで出力しますから、これをバスに送ると基本クロックとして利用することができません。しかし、私の経験からいいますと、このCPUのクロックをインターフェース回路でどうしても使わなければならないという場合はほとんどないようです。

逆にいえば、CPUの処理と同期してインターフェース側の処理をしなければならないようであれば、そこにもう一つのマイクロプロセッサを使ったほうがよいのだともいえます。マイクロプロセッサはコンピュータとしても使えますが、一つの機器の専用回路としても使えますから、インターフェース回路のなかにも可能ならばふんだんに投入したらよいと思います。

さて、一つのクロック周期を T サイクルと呼ぶと、マイクロプロセッサの基本操作は、この T サイクルをいくつか使うことによって構成できます。この基本操作をマシンサイクル（または M サイクル）といいます。

一つのマシンサイクルのなかで実行される基本操作をバス側からみて分類すると次のようになります。

- (1) メモリの読出し (memory read)
- (2) メモリへ書込み (memory write)
- (3) I/Oの読出し (I/O read)
- (4) I/Oへ書込み (I/O write)
- (5) 割込み受付け (interrupt acknowledge)

8080AのCPUからこの5種類の信号が独立に出るのではないのですが、8228システムコントローラを使うと、(1)～(5)の信号が得られますので、ここではCPUまわりがこの信号を発生するように構成されていることを前提にして話を進めます。参考のために、インテル社が発表している8080A、8224、

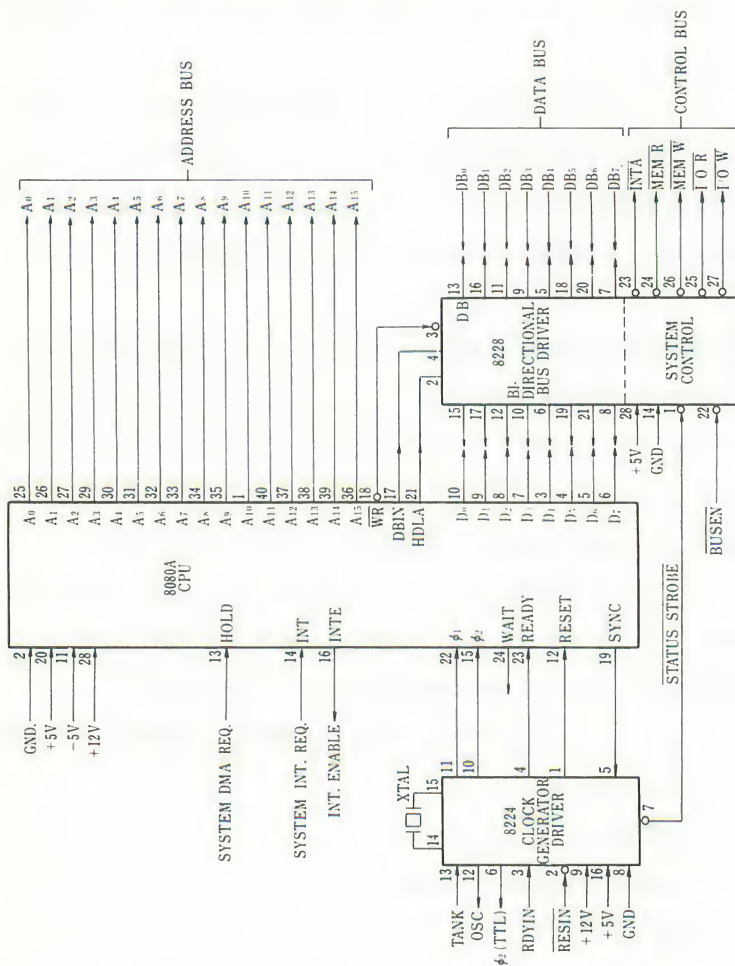


図 2.2 8080 A の CPU まわりの接続例 (インテル 8080 マイクロコンピュータシステムズ ユーザズ マニュアル)

8228の接続例を図 2.2に再掲します。

さて、8080Aはメモリに関係した命令と入出力装置に関係した命令の2種類をもっています。最初に、メモリ関係の命令の動作について考えます。

たとえば CPU が

LDA 200

という命令を実行すると、メモリの200番地の内容をCPUのAレジスタに読み出してくることになります。このときアドレスバス $A_0 \sim A_{15}$ に200が出力され、メモリ読出しのバス制御線 $\overline{\text{MEMR}}$ がロウに落ちます。だから、逆にいうと、メモリを設計するときに、 $\overline{\text{MEMR}}$ がロウに落ちたら、そのときアドレスバスに乗っている番地のメモリセルがデータバスをドライブするように設計しておかなければならないということになります。

次に、メモリへの書込みですが、CPUがたとえば

STA 300

という命令を実行すると、このときはAレジスタの内容がメモリの300番地に転送されなければなりません。このとき300がアドレスバスに乗り、Aレジスタの内容がデータバスに乗り、その後に $\overline{\text{MEMW}}$ の信号がロウに落ちます。だから、 $\overline{\text{MEMW}}$ がロウに落ちたら、そのときアドレスバスに乗っている番地のセルに、そのときのデータバスの内容を書込むようにメモリを設計しておかなければなりません。

メモリ回路の設計について、今回はふれない予定です。この程度にしておいて、かんじんのI/O読出しとI/O書込みについて、もう少し詳しく検討してみます。

いま、仮りにCPUが

IN 95

という命令を実行したとします。これは入力ポート番号95₁₀の情報をAレジスタに読み出してくる命令ですが、アドレスバスの $A_0 \sim A_7$ と $A_8 \sim A_{15}$ に重複して95が8ビットの2進数になって出力されます。ということは、入力ポートの番号は0～255までで、合計256個の入力ポートが設置できるということになります。データをセットする信号として、バス制御線のなかの $\overline{\text{IOR}}$ がロウに落ちます。この間のタイミングは図2.3のようになっています。

入力ポートからデータを読込むマシンサイクルは T_1 , T_2 , T_3 , T_4 という

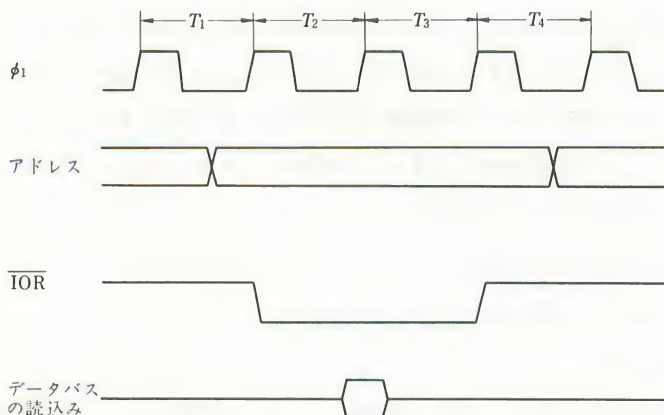


図 2.3 8080A の I/O 読み込みのタイミング

4 つの T サイクルから構成されていて、まず、 T_1 サイクルの途中で 95 が 2 進数の形でアドレスバスに出てきます。 $\overline{\text{IOR}}$ は T_2 サイクルの最初にロウに落ち、 T_3 サイクルの終わりにハイに戻ります。この間、CPU は T_3 サイクルの始まりのところでデータバスの内容を A レジスタに読み込みます。前に説明したデータのセットアップ時間とホールド時間を見込んで、この前後でデータはバス上で安定していなければなりません。

大ざっぱに考えて、CPU がデータを読み込む 1 サイクル前に $\overline{\text{IOR}}$ が出、1.5 サイクル前にアドレスが出ると考えておけばよいでしょう。

もし、95 の入力ポートの情報がフリップフロップにラッチされた性質のものであったとすると、図 2.4 の回路によって、CPU が

IN / 5 F

の命令を実行したときに、この 8 個のフリップフロップの内容が A レジスタのなかに読み込まれることになります (勿論、CPU がデータを読み込んでいるときに、このフリップフロップの内容を変化させてはいけません)。10 進数の 95 は 2 進数に変換すると、

0101 1111

となりますから、 A_7 と A_5 に NOT 回路をつけ、それらを \overline{IOR} と一緒にして SN74367 の禁止信号を作ります。

SN74367 は後の 2.5 節でもう少し詳しく述べますが、3 状態のゲートが 6 個入った IC です。パッケージのピン配置は図 2.5 のようになっています。6 個のゲートは 4 個と 2 個にわかれて使用でき、それらは独立に禁止できるようになっています。禁止入力をハイにすると、ゲートは第 3 状態

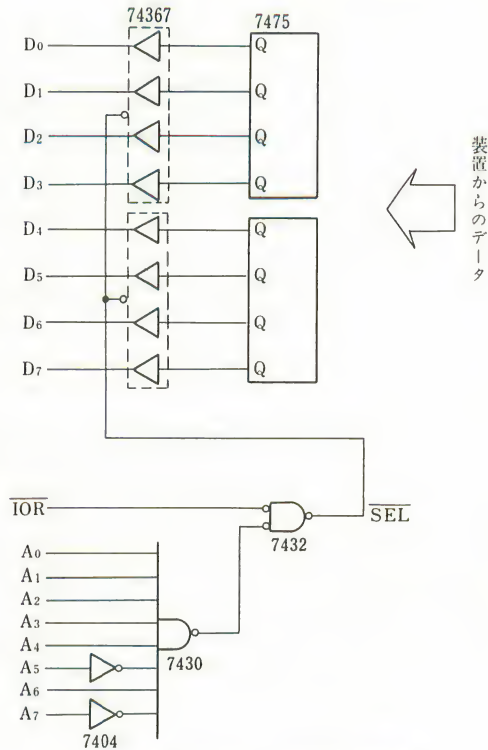


図 2.4 入力ポート / 5 F の構成

り離されます。禁止入力をロウにするとその素子がバスをドライブします。このとき、SN74367 はデータがそのままバスに出ていきますが、SN74368 はデータの 1 と 0 が逆になって出力されます。便利な IC で、バスを作るときによく利用されます。

こういったゲートを通る間での信号の遅れは、マイクロプロセッサのマシンサイクルの長さに比べればはるかに短いのので、 \overline{IOR} が出ている間にフリップフロップの内容を変化させさえしなければ、データは安定して CPU に読み込まれます。

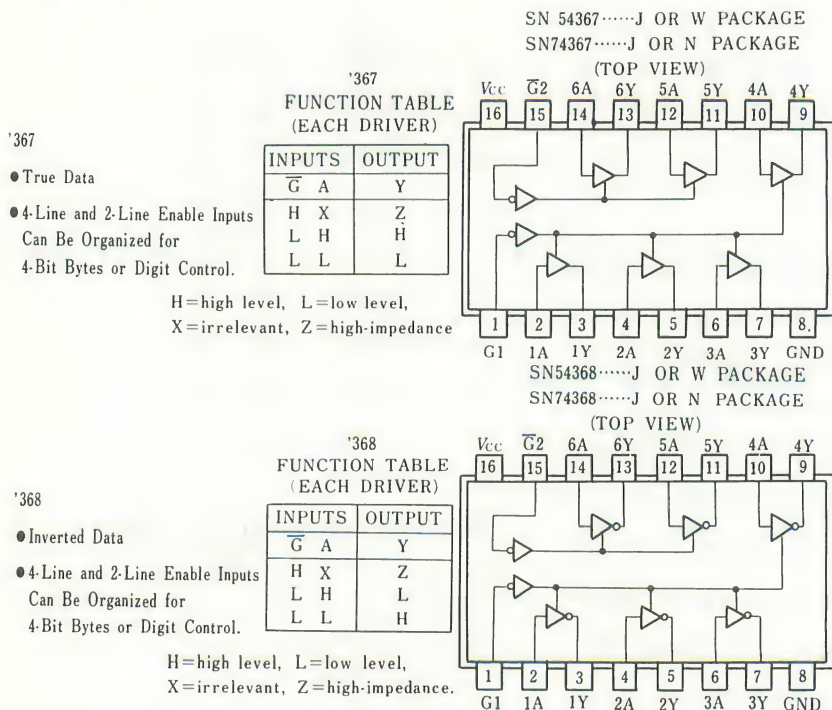


図 2.5 SN 74367 (368) のピン配置 (TI社資料)

だから、逆にいえば、インターフェース回路を設計するに当たって、CPUが入力ポートからデータを読み込んでいるときに、その内容を変化させてはいけない、ということになります。こちらのほうが実際の問題を解決するに当たって設計のポイントになります。

次に、マイクロプロセッサが出力ポートに情報を送り出す場合について考えます。たとえば、いま CPU が

OUT /57

という命令を実行したとします。このとき、アドレスバス $A_0 \sim A_7$ と同じく $A_8 \sim A_{15}$ に /57 が出力され、データバスにそのときの A レジスタの内容が出

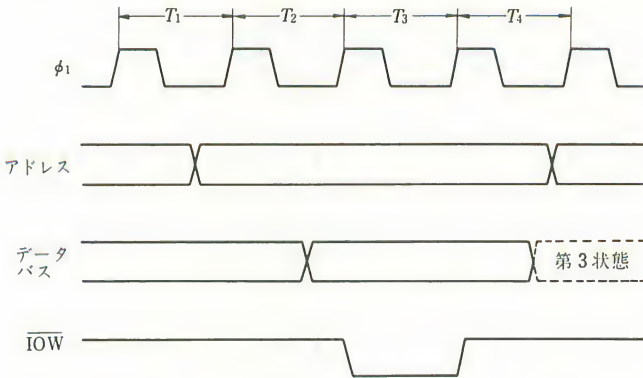


図 2.6 8080A の I/O 書込みのタイミング

力され、 \overline{IOW} 信号がロウに落ちます。

このときの信号のタイミングは図 2.6 のようになっています。アドレスは T_1 サイクルの中頃に、データは T_2 の中頃にバスに出てきます。 \overline{IOW} の制御信号は T_3 サイクルに出てきます。インターフェース回路のなかで信号の遅れが小さければ、 \overline{IOW} の前縁 (leading edge) の時点でデータをセットすることができます。勿論、後縁 (trailing edge) でもかまいません。どちらがよいかということになれば、時間的に余裕のある後縁のほうがよいといえます。

いま仮りに、A レジスタの内容を SN7475 タイプの D フリップフロップにラッチするとします。図 2.7 のように設計すれば、A レジスタの内容がインターフェース回路に送られてきます。このようなタイプの出力回路を出力ポート (output port) といいます。

SN7475 は D フリップフロップを 4 個もった 16 ピン IC ですが、クロック CLK は 2 個のフリップフロップに共通になっています (図 2.8)。クロックがハイのとき、入力 D がそのまま出力 Q にでてきますが (\overline{Q} は入力を否定したもの)、CLK がロウに落ちたときに、そのときの入力の状態がフリップフロップにラッチされます。だから、このフリップフロップは CLK の立下り

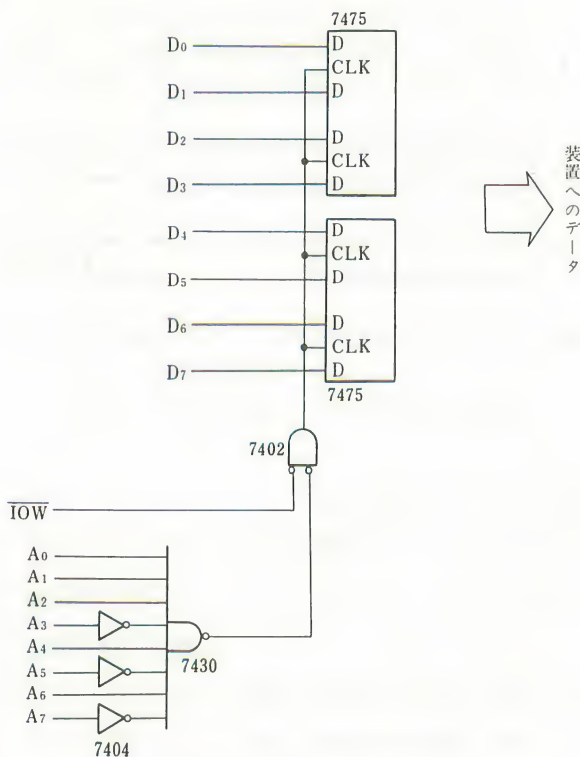
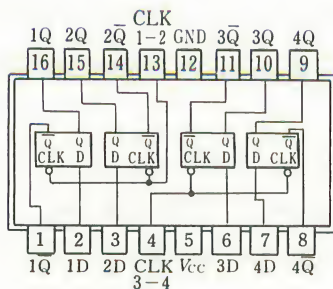


図 2.7 出力ポート／57の構成



真理値表 (Each Latch)	
I_n	I_{n+1}
D	Q
1	1
0	0

図 2.8 SN7475のピン配置と真理値表 (TI 社資料)

で情報を取り込むと考えなければなりません。図 2.7 のように回路を組めば $\overline{\text{IOW}}$ の後縁でデータのセットが行なわれます。

もし、インターフェース回路において、パルスが必要なときは図 2.9 および図 2.10 の回路でパルスを作ることができます。まず、図 2.9 の場合ですが、この場合は、A レジスタに 1 をセットしておいて、OUT、 $\nearrow A 5$ という命令を実行すると、対応する出力線のところにパルスが出てきます。たとえば、 $\overline{\text{PA5}}_6$ の端子だけにパルスを出したいときは

```
MVI A,  $\nearrow 40$ 
OUT  $\nearrow A 5$ 
```

という命令を実行すればよいことになります。もし、

```
MVI A,  $\nearrow \text{FF}$ 
OUT  $\nearrow A 5$ 
```

とすると、8 本の全部の端子に同時にパルスが出てきます。もし、ポート番号に余裕があれば、図 2.10 の回路によってパルスを作ることができます。この回路を使うと、CPU が

```
OUT  $\nearrow B 2$ 
```

という命令を実行したとき、 $\overline{\text{PB2}}$ の端子にパルスが出てきます。前のように A レジスタにデータをセットするわずらわしさはなくなりましたが、一時

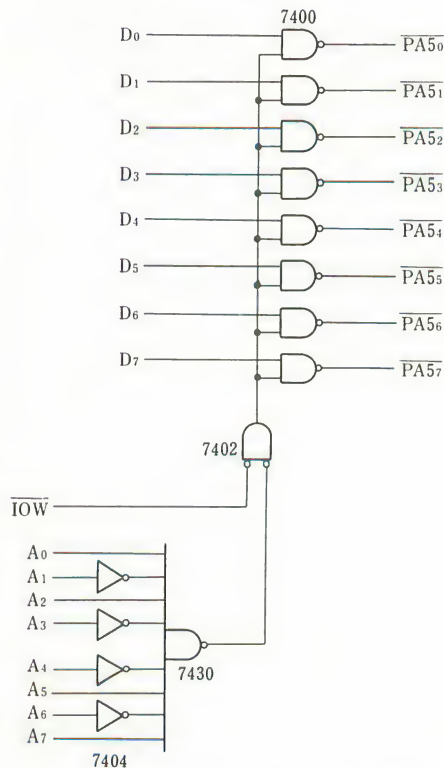


図 2.9 $\overline{\text{IOW}}$ と同じ幅のパルスを得る回路

に、一つのパルスしか出ないことになります。SN7442はBCDコードを0～9までにデコードするICです(図2.11)。以上の手順を整理すると表2.1のようになります。

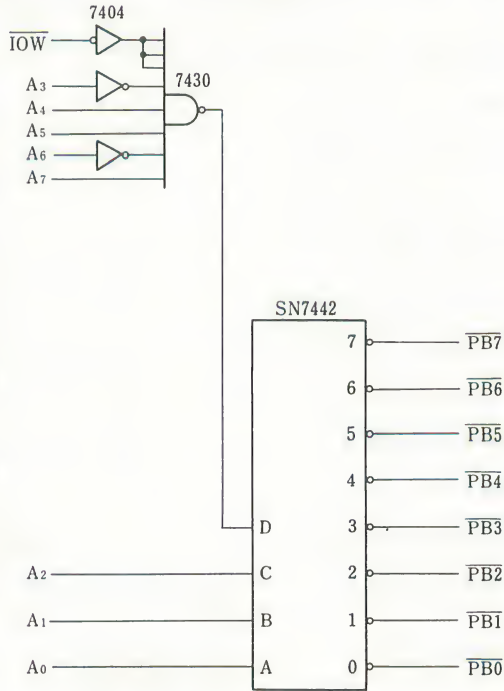


図 2.10 OUT 命令だけでパルスをつくる回路

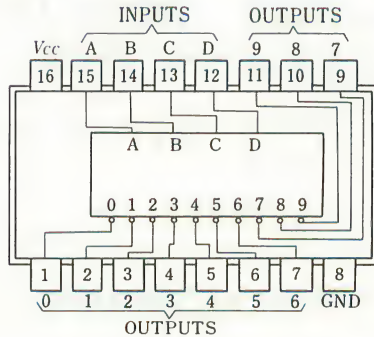


図2.11 (a)
SN7442のピン配置

BCD INPUT				DECIMAL OUTPUT									
D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	0	1	1	1	1	1	1	1
0	1	0	0	1	1	1	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	1	1	0	1	1	1	1
0	1	1	1	1	1	1	1	1	1	0	1	1	1
1	0	0	0	1	1	1	1	1	1	1	0	1	1
1	0	0	1	1	1	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

図 2.11(b) SN7442の真理値表(TI 社資料)

表 2.1 メモリとI/O関係のバス制御信号の分類

	種別	信号名	内 容	アドレスバス	データバス
1	メモリ	$\overline{\text{MEMR}}$	メモリ読出し	メモリ番地	指定されたメモリセルがドライブ
2		$\overline{\text{MEMW}}$	メモリ書込み	$A_0 \sim A_{15}$	マイクロプロセッサがドライブ
3	I/O	$\overline{\text{IOR}}$	I/O 読出し	ポート番号	I/O ポート側がドライブ
4		$\overline{\text{IOW}}$	I/O 書込み	$A_0 \sim A_7, A_8 \sim A_{15}$	マイクロプロセッサがドライブ

48 第2章 マイクロプロセッサのバスの実例

これまでの説明からわかるように、メモリと I/O といっても、それらを区別する要素は物理的なものではありません。ただ違ったバス制御線にパルスが出てくるというだけのものです。だから、たとえばメモリ容量が大きくないときには入出力機器のポートをメモリ領域に割りつけることができます。たとえば、図 2.4 のかわりに図 2.12 のようにすれば、CPU が

LDA / FFFA

という命令を実行したときに、このフリップフロップの状態が A レジスタに入ります。CPU 内の情報を入出力機器に送り出す場合もまったく同じです。図 2.13 は CPU が

SHLD / FCFE

を実行したときに、HL レジスタの内容が 16 ビットの D フリップフロップにセットされる回路です。参考にして下さい。

このように、入出力機器をメモリ領域内に設定することをインテル社では memory mapped I/O と呼んでいます。これは、8080A の命令体系がメモリ関係に豊富なのに、一方で、I/O 関係には

IN ポート番号
OUT ポート番号

の 2 種類しかないというアンバランスからきています。入出力装置を I/O 側に接続すると、情報は A レジスタを介してのみしか転送されませんが、メモリ領域に割りつけると、どのレジスタとも情報交換ができることになったり、16 ビットの転送も可能になります。だから、必要なメモリが 64K バイトにならないことがはっきりしているときには、入出力装置をメモリの番地に割りつけるとよいでしょう。

しかし一方で、IN と OUT の命令は 2 バイト命令なのに、STA と LDA などの命令は番地部を含めて 3 バイト命令になりますから、入出力装置をメモリ領域に割りつければ、何でも必ずよくなるというものではありません。これ

はI/O側の操作がアドレスを半分しか使っていないのに、メモリの場合は $A_0 \sim A_{15}$ の16本のアドレスをフルに使っているからです。このことは図2.4と図2.12のアドレスデコード部を見てもわかります。

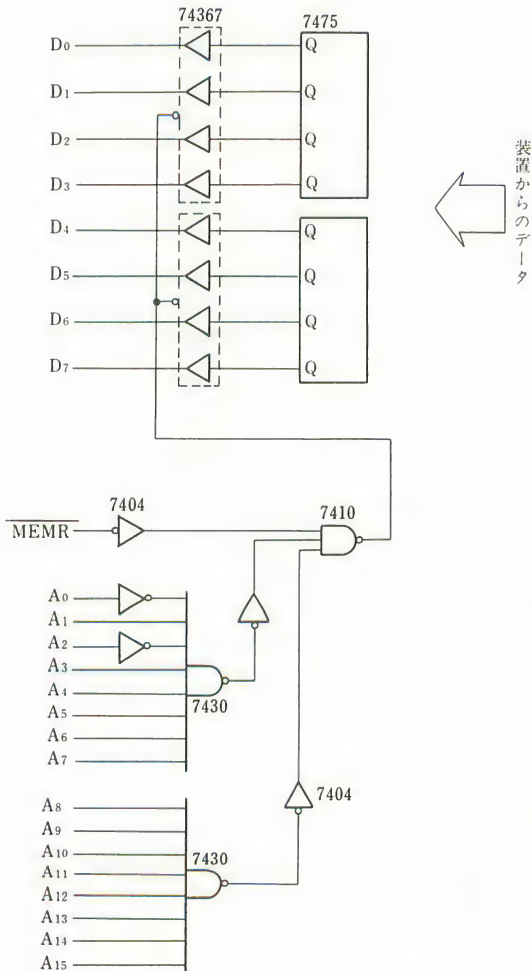


図 2.12 メモリの / FFFA 番地に割り当てられた入力ポート

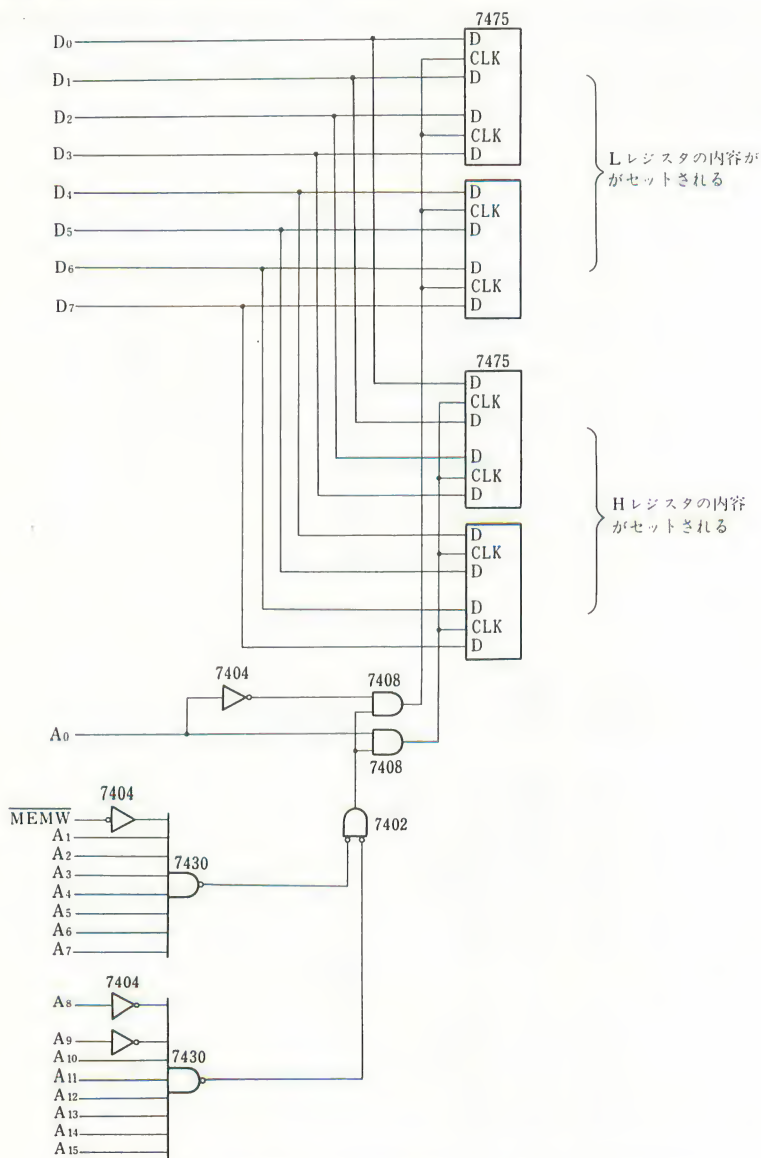


図 2.13 SHLD/FCFE を実行したときに HL レジスタの内容がセットされる出力ポート

また、入出力装置をメモリ領域に割りつければ、その番地はメモリとして完全に使用不可になるという潜在的なマイナスもあります。

このように長短あるわけですが、これらはどちらにしようとしたことではありませんから、ここでは入出力装置はすべて I/O 側につけるものと仮定して話を進めることにします。

最後に、8080 A の割込みについて述べます。割込みというのは、CPU の正常な実行ルーチンを一時的に中断して、他の場所に格納されている命令群を実行し、その後に、中断したプログラムルーチンの実行を再開するというものです。

8080 A の割込みを可能にするために EI (Enable Interrupt) という命令があります。8080 A はリセットスタートした時点では、割込みを受け付けない状態になっていますから、プログラムのどこかに EI の命令を設定して、割込みを受け付ける状態にしなければなりません。

さて、EI 命令を実行することによって、CPU が割込みを受け付ける状態になっていたとします。このときに、割込み要求線 INT をハイにあげると、これは命令を実行している 8080 A の最後の T サイクルで認知され、その次のサイクルで $\overline{\text{INTA}}$ がロウに落ちます。このときの信号のタイミングは 図 2.14 のようになっています。信号の性質は $\overline{\text{MEMR}}$ と $\overline{\text{IOR}}$ のときと同じですが割込み要求したインターフェース回路はこの $\overline{\text{INTA}}$ によって、一つの命令を

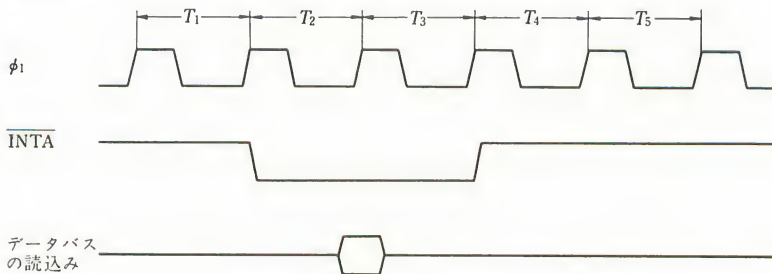


図 2.14 $\overline{\text{INTA}}$ のタイミング

52 第2章 マイクロプロセッサのバスの実例

CPU に送り込まなければなりません。なぜそんなことをしなければならないかについて考えてみます。

マイクロコンピュータの命令の実行の過程において、命令は一つ一つ順々に実行されます。この命令はメモリのなかに格納されているので、マイクロコンピュータとしては、命令実行の最初のマシンサイクルで、必ずメモリに命令を取りにいかねばなりません。このサイクルをとくに M_1 サイクル、またはマシンサイクル 1 といいます。

マイクロプロセッサは次に実行する命令のアドレスをプログラムカウンタのなかにもっています。 M_1 サイクルにおいて、このプログラムカウンタがアドレスバスに出力され、 $\overline{\text{MEMR}}$ のバス制御線がロウに落ち、メモリ読出しが実行されます。これでプログラムカウンタで指定されたメモリの内容が CPU に読み込まれ、命令として解釈され、その後、その命令に応じた処理が行なわれます。

一つの命令がプロセッサに読み込まれると、プログラムカウンタは適当にインクリメントされ、その次に実行する命令がしまわれている番地を指します。

以上の手順を例によって説明しましょう。いま仮りに、メモリに次のようなプログラムが格納されていたとします。

0 番地	LXI SP, 100
3 番地	EI
4 番地	MOV B, A
5 番地	MOV C, B
6 番地	HLT

マイクロコンピュータの電源をオンにすると、リセット信号が入り、プログラムカウンタはゼロにクリアされ、0 番地の命令から処理が開始されます。

まず、アドレスバスにゼロが出力され、 $\overline{\text{MEMR}}$ がロウに落ちメモリ読出しを行ないますが、0番地、1番地、2番地には3バイト命令のLXI SP、100が格納されているので、これが実行されると、スタックポインタに100がセットされ、プログラムカウンタは3になります。

次に、アドレスバスに3が出力され、 $\overline{\text{MEMR}}$ がロウに落ち、メモリからEIを読み出してきて、これを解釈し、CPUを割り込み受付可能の状態にし、プログラムカウンタを4にします。

次に、アドレスバスに4が出力され、 $\overline{\text{MEMR}}$ がロウに落ちメモリからMOV B, Aを読み出してきて、プログラムカウンタを5にします。

さて、普通ならば、ここでアドレスバスに5が出力され、5番地の命令が取り込まれるはずですが、MOV B, Aの命令を実行しているときに、INT線がハイに上がって割り込み要求がきたとします。この要求はMOV B, Aの命令の処理の最後で受け付けられます。

このとき、プログラムカウンタは5になっていますが、そのまま命令の実行を続けるわけにはいきません。次の M_1 サイクルにおいて、アドレスバスには一応プログラムカウンタの5が出力されますが、この情報は無視されます。CPUは $\overline{\text{MEMR}}$ をロウに落とすかわりに、 $\overline{\text{INTA}}$ を $\overline{\text{MEMR}}$ とほとんど同じタイミングでロウに落とします。アドレスバスに5が出ても、 $\overline{\text{MEMR}}$ がロウに落ちませんから、この M_1 サイクルではメモリはバスに情報を供給しません。

以上の理由によって、割り込み要求を出した機器が、 $\overline{\text{INTA}}$ に合わせてバスに命令を送り込まなければならないのです。このときに送り込む命令は

RST n

という命令ですが、 n は0～7のいずれかです。

CPUは普通の $\overline{\text{MEMR}}$ によって命令を読み出してきたときは、プログラムカウンタをインクリメントしますが、 $\overline{\text{INTA}}$ によって機器から命令を取ったときはプログラムカウンタの内容をインクリメントすることはできません。

今の例の場合のように、4番地の命令を実行しているときに割込みが入ったときは、プログラムカウンタは5になっていますが、 $\overline{\text{INTA}}$ によってRST n を読み込んだときにプログラムカウンタをインクリメントすると、5番地の命令をまだ実行していないのに6となってしまうと、5番地の命令がスキップされてしまうからです。これが割込み命令の特殊なところですよ。

さて、RST n の命令がCPUによって実行されると、そのときのプログラムカウンタの内容（この場合は5）が、スタックポインタで指定されるメモリで格納されます。この場合、スタックポインタは100になっていますから、

98番地 0000 0101 (下位アドレス)
99番地 0000 0000 (上位アドレス)

というようにプログラムカウンタの内容がスタックに退避されます。

プログラムカウンタには表2.2に示した飛び先番地が新しくセットされま

表2.2 RST 命令の飛び先番地

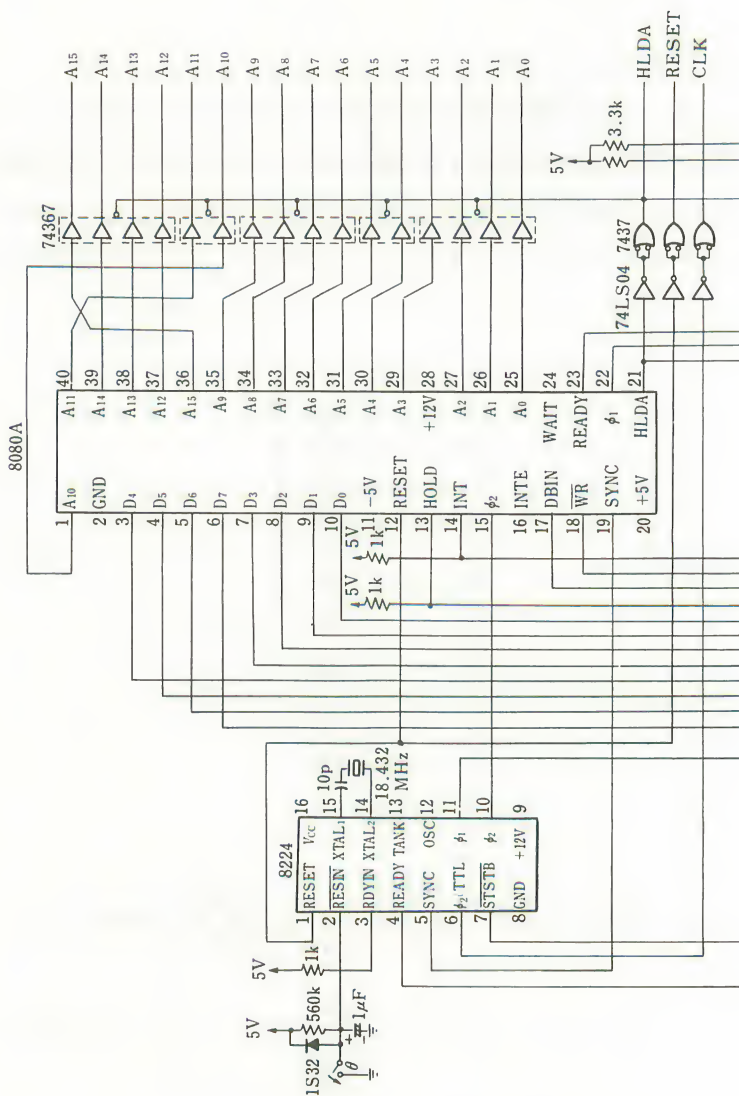
命 令	n	飛び先番地	16進番地
RST 0	000	0	0000
RST 1	001	8	0008
RST 2	010	16	0010
RST 3	011	24	0018
RST 4	100	32	0020
RST 5	101	40	0028
RST 6	110	48	0030
RST 7	111	56	0038

す。ということは、結局、その番地から割込み処理プログラムが開始されるということの意味します。割込み処理が終わったらRETという命令を実行すれば、中断したプログラム（この場合5番地以降のプログラム）が再び実行されます。

RST n の命令をデータバスに入れるときのビットパターンは、

$D_7D_6D_5D_4D_3D_2D_1D_0$
1 1 × × × 1 1 1

という形になっていて、 $D_5D_4D_3$ の×××というところに n を2進数に変換したものを書き込みます。たとえば、RST 5は



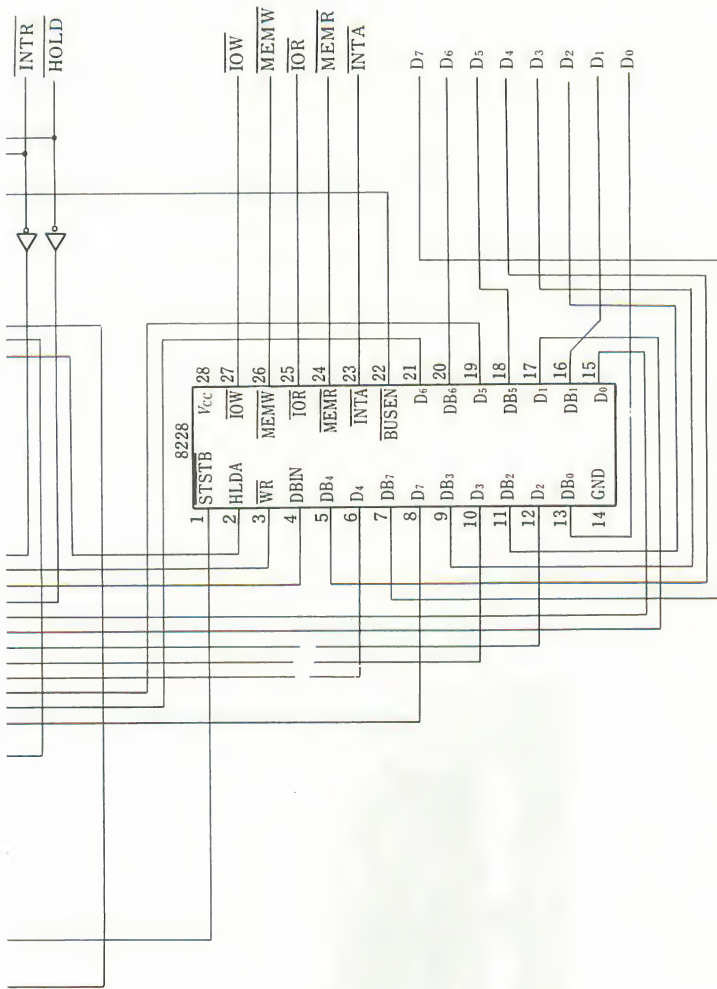


図 2.16 8080A CPU カードの設計例

割込みを受け付ける状態になっていれば、命令の終りでこれと認知し、その次のマシンサイクルで $\overline{\text{INTA}}$ をロウに落とします。このタイミングで SN74367 のゲートを開くと、データバスに 1110 1111 の信号が乗り、RST 5 の命令が実行されます。このとき CPU は中断したプログラムカウンタをスタックにしまい、そしてコントロールは /28 番地に飛びます。

/28 番地以降に必要なプログラムを書いておくと、そのプログラムが実行され、最後に、RET 命令を実行することによって、中断されたプログラムが再開されることになります。割込み要求フリップフロップは $\overline{\text{INTA}}$ の後縁でリセットします。

割込みを要求する入出力機器が 1 個でなくて複数個ある場合は図 2.15 のように単純にはいきません。それは、2 以上の割込み要求が同時に発生したときに、どちらを優先するかということをあらかじめ決めておかねばならないからです。この優先度の決め方には種々の方法がありますが、それらの詳細は後の機会にゆずることにします。

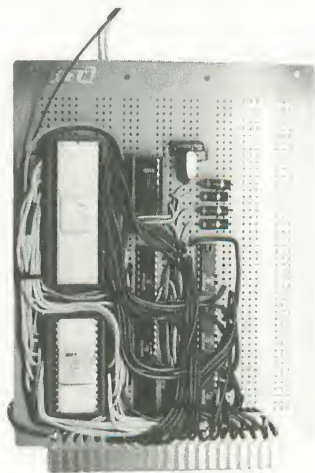


写真 2.1 8080A CPU カード
の製作例

以上で、8080 Aのインターフェース回路を設計するに当たって、 $\overline{\text{IOR}}$ と $\overline{\text{IOW}}$ と $\overline{\text{INTA}}$ に着目すればよいことがわかったと思います。この他に、DMA 操作について、 HOLD と HLDA 信号を使いますが、これは後に実例について述べるときに説明します。

参考のために、図 2. 16に8080 Aの CPU カードの設計例を示しました。写真 2. 1はユニバーサル基板の上に組んだ実例です。

2.3 8085のバス

そうこうしているうちに、1977年に入って、インテル社より8085の発売がアナウンスされました。命令体系は8080 Aとほとんどかわりませんが、処理速度が向上し、同時に、8080 Aと8224と8228が分担していた役割を一つのCPU のチップのなかに集積することに成功しました。ここでは8085のバスを8080 Aの場合との相違点に着目して調べてみることにします。

まず、クロックですが、8080 Aが $t_{cy} = 500\text{nsec}$ だったのにたいして、8085では330 nsec になっています。これで処理速度が約 1/3 向上したことになります。クロックはTTL レベルで出力されますから、これをバスに送ることは可能です。

メモリ関係は省略することにして、インターフェース関係ですが、これには

$\overline{\text{RD}}$ read
 $\overline{\text{WR}}$ write
 $\overline{\text{IO}/\overline{\text{M}}}$ I/O
 $\overline{\text{INTA}}$ interrupt acknowledge

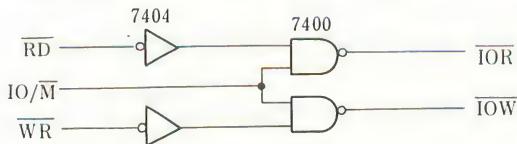


図 2. 17 8085の制御信号のデコード

の4つの制御線がCPUから独立に出力されています。この制御線を図2.17のようにデコードして、8080Aの制御線と同じ信号を作ることができます、これら4本の線をそのままバスに送ることもできます。バス制御線を8080Aの場合と同じにしたほうがよいかどうか、いまのところははっきりわかりません。

まず、入出力装置から情報をCPUに読み込む場合ですが、これは図2.18のように、 T_1 サイクルでアドレスが出力され、 T_2 サイクルのはじめに \overline{RD} がロウに落ち、 T_3 の前半でデータがCPUに読み込まれ、その後に \overline{RD} がハイに上がるということになります。 \overline{RD} がロウに落ちてから、データがCPUに読み込まれるまでに400nsecぐらいの時間がありますから、図2.19の回路で余裕のある情報転送ができます。

図2.19の回路において、CPUが

IN /DB

という命令を実行すると、 $IO/\overline{M}=1$ で \overline{RD} がロウに落ち、そのときアドレスバスの $A_8 \sim A_{15}$ に

1101 1011

が出力されますから、このときにかぎり \overline{SEL} がロウに落ち、SN74367のゲ

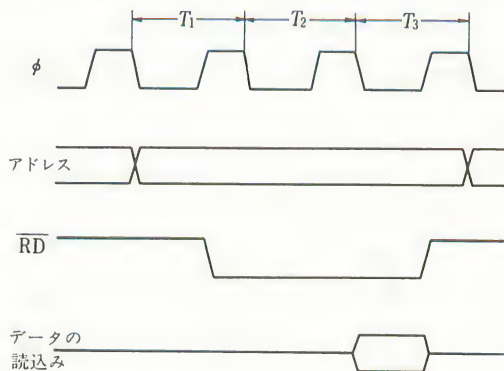


図 2.18 データ読込みのタイミング

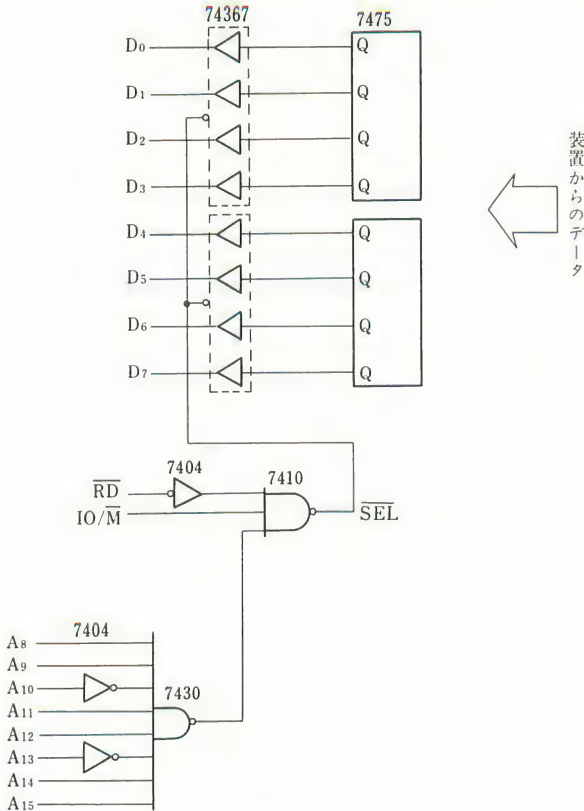


図 2.19 8085における入力ポート／DBの回路

ートが開き、D フリップフロップの内容がバスに出てきます。

次に、A レジスタの情報の出力操作ですが、このときの信号のタイミングは図 2.20 のようになります。アドレスは T_1 サイクルのはじめに出てきますが、データが T_2 サイクルのはじめでないと出てきません。これは 8085 のアドレスバスが上位 $A_8 \sim A_{15}$ の 8 本しかなく、下位の $A_0 \sim A_7$ がデータバスから出てくるという特殊事情によります。

T_1 サイクルのデータバスにはアドレス $A_0 \sim A_7$ が乗っています。 \overline{WR} の信号は T_2 サイクルのはじめにロウに落ちますが、このときはデータはバスの上で

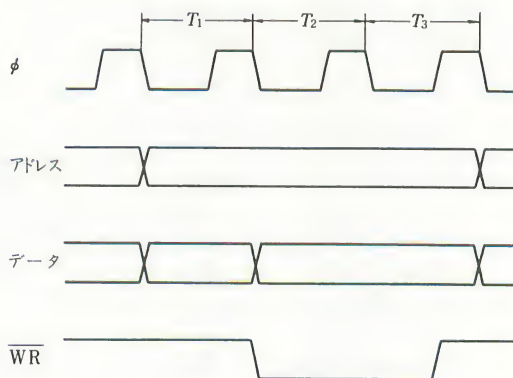


図 2.20 8085におけるデータの書き込み
のタイミング

安定していません。したがって8085の場合、必ず $\overline{\text{WR}}$ の後縁でデータをセットするようにします。

図 2.21は、CPU が命令

OUT /6 D

を実行したときに、そのときのAレジスタの内容がフリップフロップにセットされる出力ポートです。 $\overline{\text{WR}}$ の制御線の後縁を使ってデータをセットしているところに注意して下さい。これを前縁にすると正しいデータのセットは行なわれません。

次に、割込み処理についてですが、8085はアドレスバスを節約しただけピンに余裕があり、それだけ割込み機能が強化されています。

まず、CPUの状態いかににかかわらず、信号線をロウからハイに上げると、コントロールが必ず/24番地にジャンプするという割込みがあります。これをTRAP またはマスク不可能な割込みといいます。

これは、たとえばAC電源が事故などで切れたときに、それを検知して、DC電源が落ちる前に何らかの処理をする、というように使います。

TRAP の他に

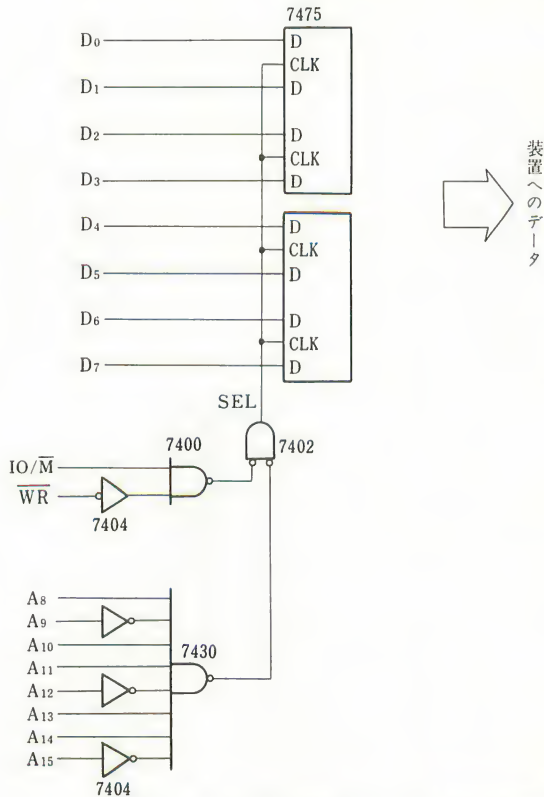


図 2.21 8085の出力ポート / 6 D の回路

RST 7.5
RST 6.5
RST 5.5

という飛び先の定まった割込み端子があります。このなかでRST 7.5がもっとも優先度が高く、次にRST 6.5、最後にRST 5.5という順序になっています。割込みの検知はRST 6.5とRST 5.5がレベル検知で、RST 5.5は立上り検知です。したがって、RST 6.5とRST 5.5はインターフェース回路のなかで割込み要求をラッチする必要がありますが、RST 7.5はパルスを入れればよいことになります。

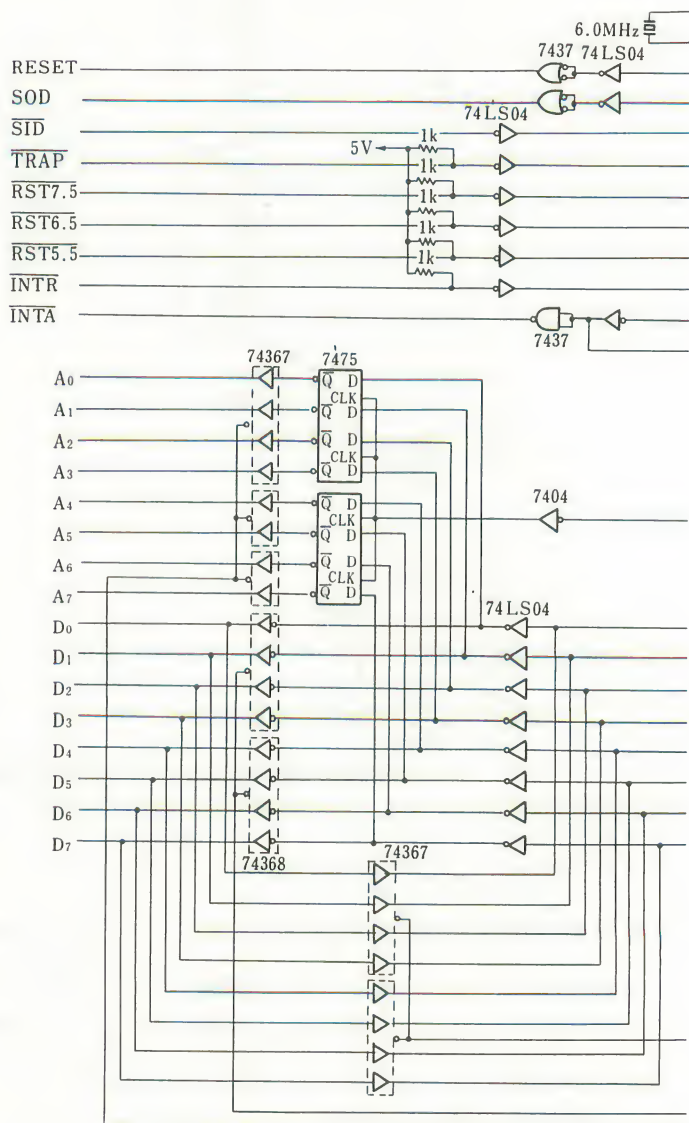
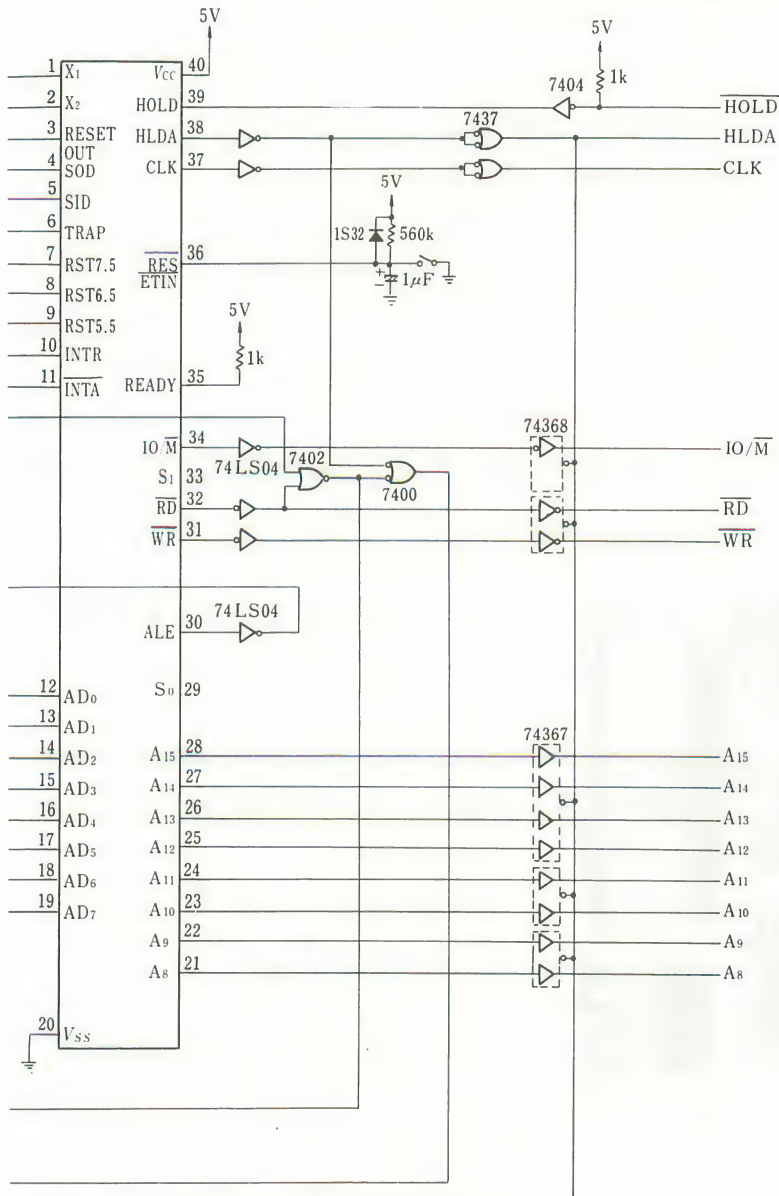


図 2.22 8085CPU



カードの設計例

この3つの割込みが入ると、CPUは自動的にRST命令を作って、

RST 7.5のとき／3C番地に、

RST 6.5のとき／34番地に、

RST 5.5のとき／2C番地にコントロールを移します。

このとき、インターフェース回路として特別に処置することはありません。最後に、8080Aの場合とまったく同じタイプのINTRと $\overline{\text{INTA}}$ の端子があります。 $\overline{\text{INTA}}$ は $\overline{\text{RD}}$ とほとんど同じタイミングで出されますので、これを使って、インターフェース回路でRST n という命令をバスに送り込みます。これについて繰り返す必要はないでしょう。

参考のために、図2.22に8085のCPUカードの設計例を示しました。写真2.2はそれをユニバーサル基板の上に組んだ実例です。

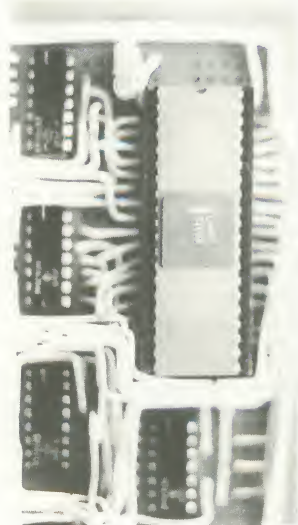


写真2.2 8085CPUカード
の製作例

2.4 バスを作るときの注意

実際にマイクロコンピュータを作るに当たって、インターフェース回路の動作原理を理解すれば、次は、その原理を物理的に実現する方法を考えなければなりません。もし、考えているマイクロコンピュータシステムが試作的なシステム、あるいは研究用で量産しないものならば、おそらくほとんどのインターフェース回路はユニバーサル基板の上に組み立てられることになるでしょう。

これまで市販されているユニバーサル基板を調べてみると、両面独立で44極のものと56極のものが多く販売されているようです。少々無理

すれば100極のものも入手することができます。

布線はハンダ型とラッピング型がありますが、ハンダ型のほうが一般に安価のようです。私の研究室では小規模のテストシステムのときに44極、実験装置などを制御するようなときでかなり実用的な性格の強いときに56極、メモリ容量の大きいシステムのときに100極のコネクタを使用することになっています。基板上の配線はほとんどの場合ハンダづけします。

写真2.3は44極のユニバーサル基板を使用した例です。写真2.4は56極のユニバーサル基板を使用したもの、写真2.5は100極のユニバーサル基板の使用例です。

コネクタのピンの配線ですが、これについて、いろいろの方法を試みてきましたが、現在では、一般にマザーボードとか共通母線方式といわれている方法を採用しています。これはコネクタをいくつか並べて、その対応するピ

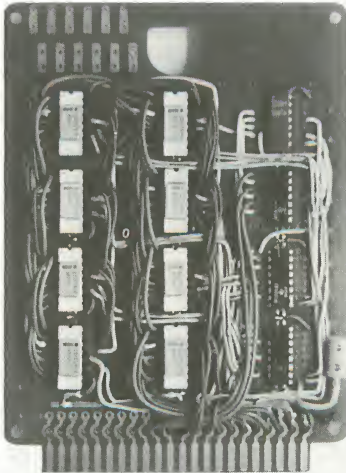


写真2.3 44極ユニバーサル
基板を用いて製作し
たメモリカード
(2102A-4)

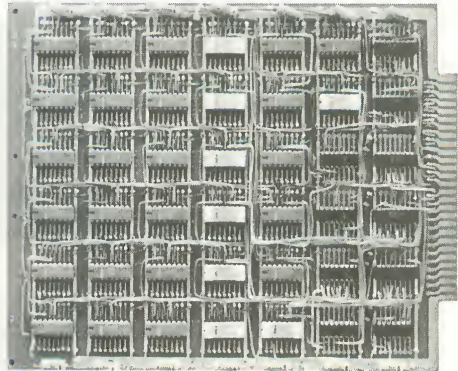


写真2.4 56極ユニバーサル基板を
用いて製作した4KBメモ
リカード (2102A-4)

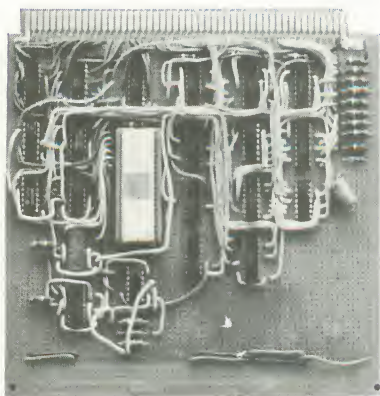


写真 2.5 100極ユニバーサル
基板の使用例

ンをすべて横に連結してしまうという
ものです。写真 2.6 は溶ダータイプ
のコネクタを手作りでマザーボード化
したものです。安価ですが、手間がか
かります。経済的な余裕があれば、写
真 2.7 のようなプリント基板に布線し
たものを購入したほうがよいでしょう。

この方式の良い点は信号が極の位置
によってすべて決まってしまうので、
あれやこれや考える必要がない点です。
これはシステムを増強したり、改良し

たりしようとするときにコネクタの布線を変更しないでよいという利益にな
ります。しかし、逆に考えれば、すべての信号がすべての基板で必要なわけ

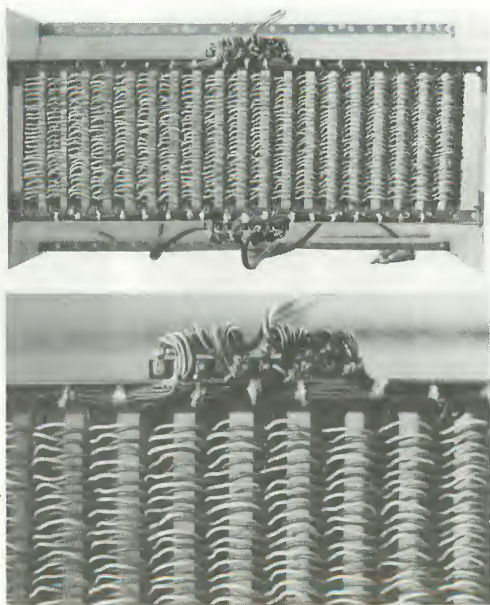


写真 2.6 ソルダータイプのコ
ネクタを手作りでマザ
ーボード化した例

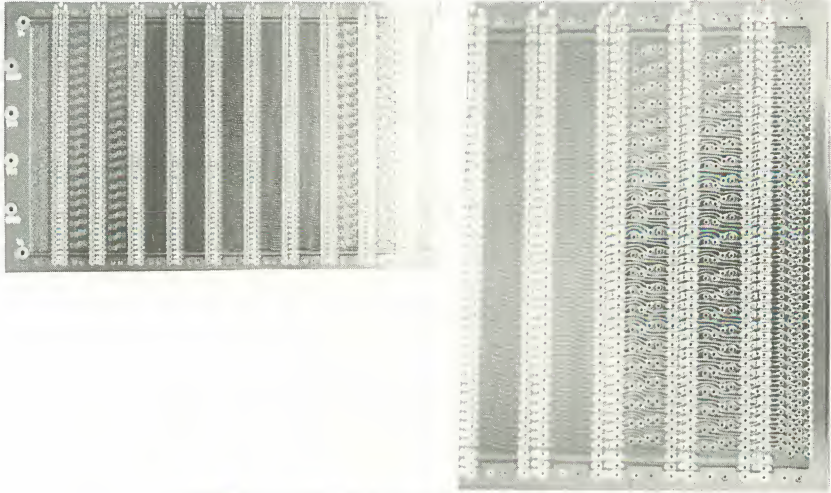


写真 2.7 市販されているマザーボードの例

ではないから、資源が遊んでいて、有効に利用される割合が低いともいえま
す。

そのどちらをとるかですが、私のこれまでの経験からいいますと、一度動
き出したシステムに、後から手を加えることにたいして、相当大きい精神的
な抵抗があるので、コネクタの布線の変更はなるべくしないほうがよいとい
うことになり、マザーボード方式に軍配をあげざるを得ないことになります。
実際、正常に動作しているシステムを改良しようとして手を入れ、それで動
作をおかしくしてしまったというようなこともないわけではなく、このとき
の精神的なショックを考えに入れば、コネクタの極数を増やすなど安いも
のだといえます。

共通布線方式のもう一つの利点は、カードの位置がスロットルのどこに入
れてもよいので、試験する基板を他のカードから離せば、たとえばシंकろ
スコープのプロブを直接入れることができ、特別の引き出しカードが不用
になり、より自然な状態でテストができます。

70 第2章 マイクロプロセッサのバスの実例

この方式を採用した場合、一つのインターフェース回路が一つのカードのなかで閉じるように設計されなければなりません。カードからカードに配線が走るようになると、数の限られた共通バスを使用しなければならないので感心しません。逆にいえば、インターフェース回路の設計が基板の大きさから影響を受けるということになります。

たとえば、ICが1枚の基板の上に乗らないので、インターフェース回路の機能を落として、ソフトウェアに負担をかけるというようなこともあるかも知れません。

バスのドライバとレシーバの IC をどれにするかという選択は、かなり重要な問題です。

一般に、MOS の LSI は電流をあまり大きくとれませんので、直接に母線をドライブすることができません。ごく小規模のシステムで、バスに接続される MOS IC が、たとえば10個以内というようなことがはっきりしている場合であれば、これは直接 CPU の出力線を各 IC に接続することができます。

しかし、そういう場合はむしろまれであって、多くの場合、出力ゲートの電流を増強し、バスに接続できるカードの数を増加させなければなりません。

バスのドライバとレシーバ回路として、いろいろな IC が発売されていますが、本格的なシステムでないかぎり、ドライバとして SN74367（または SN74368）程度のクラス、レシーバとしてローパワーショットキータイプを使用するのが普通ようです。TI の規格によりますと、SN74367 のシンク電流は

$$I_{OL}=32\text{mA}$$

となっています。一方で、LS タイプの入力端のソース電流は、普通

$$I_{IL}=-0.4\text{mA}$$

となっていますから、単純な計算で

$$32 \div 0.4 = 80$$

となって、80個のローパワーショットキータイプのゲートがドライブできるということになります。

ドライバとしてインテル8216、ナショナルセミコンダクタ DM8833、フェアチャイルド 8 T 26などを用いればバスのドライブ能力はもっと上がります。またレシーバとして、ヒステリシスをもった専用レシーバ、たとえば DM 8837などを使えば、割り算の分母が小さくなりますから、ドライブ可能数はさらに一段と大きくなります。

しかし、こういったバス専用のドライバ、レシーバ素子は価格もかなり高いので、それを使うかどうかは良く考えたほうがよいと思います。

最後に、インターフェース回路から入出力装置に送る信号線ですが、物理的な事情から、多くの場合、ケーブルの線長が数メートル程度になることが多いようです。信号線が長くなると、通常のディジタル IC の設計手法はそのまま適用できません。このような状況では、ケーブルそれ自体が負荷になって、複雑な応答を示しますから、伝送系の設計に当たっては細心の注意が必要です。

これらの点について、一般論として述べるのはかなりむづかしいので、私の研究室で使用している回路例を次章以下の必要なところで述べることにします。多くの場合、最終回路は実験的に調整した結果であって、なぜそのようにしたかというような計算式などをそえることができないのが残念です。

しかし、インターフェース回路の設計の技術において、経験がかなり重要なファクタであって、何から何まですべてが論理的に割り切れるものでないということを知ることも大切です。手をこまねいて何もしないぐらいだったら、実際に回路を作って実験してみたほうがよい、というのが私の考えかたの基本です。皆さんはどう考えますか。

さて、一般論は以上で終りにして、次章以下で、入出力装置のインターフェース回路の設計例を具体的にみていくことにします。

第3章

入出力タイプライタのインターフェース

3.1 はじめに

昭和45年に、黒沢通信工業から、オフラインの紙テープパンチとしてDR 5388データライタ（写真3.1）を1台購入しました。最近になって、マイク



写真 3.1 データライタ DR 5388
（黒沢通信工業）

ロコンピュータを作ることが多くなり、入出力タイプライタが不足してきたので、オフラインパンチなどというのんびりしたことがいっておれなくなり、急拠、オフラインパンチャをオンライン入出力タイプライタに改造することにしました。

黒沢通信工業に手紙を書いたり、電話したりして、図面を送って下さいと何度お願いしましたが、ナシのつぶてで、何の返事也没有ませんでした。やむをえず、岐阜の田舎から新幹線に乗って上京し、府中の工場まで押しかけ、設計を担当した技術者に面会を求め、やっとのことで1枚の図面を貰って帰ってきました。

帰宅して、さっそく図面を検討しましたが、いろいろ不都合な点があるこ

とに気がつきました。

第1に、このデータライタはリレー回路を使用しているが、その火花消弧回路にバリスタが使われていて、これではロジック回路に相当なノイズを出すことが予想される点です。要するに、オフラインのタイプライタだから、素子を少しばかり節約して安上りなものを作ったな、という感じがしました。

これが、もしオンラインのタイプライタならば、自分のところで設計したロジック回路が誤動作するだろうから、消弧回路にしても、もう少しがっちりした素子を入れるでしょうが、オフライン機器として、蓋をしてしまえば、それでよいというわけです。オフライン機器とオンライン機器で、こういうところまで違うものとは、このときはじめて知りました。

第2に、オフラインのタイプライタは、オフライン機器としてそれだけでまとまった機能をもたなければならないから、リレー回路でかなりやっかいなロジックが組み込まれている点です。

たとえば、紙テープを読み取っているときに、ある特定のコード、これをストップコードというが、そのコードがくると、キーボードがロックされ、読み取りがストップするというような機能があります。この動作はオフラインのタイプライタとして、いくつかの紙テープを編集したりするときに必要なのですが、一方で、マイクロコンピュータの入力装置としてみたとき、バイナリー形式の紙テープの読み込みが不可になるという、とんでもない結果をもたらします。マイクロコンピュータのバイナリーローダを作ったとき、あらゆるビットパターンの情報を読み込まなければなりません、そのうちのひとつがストップコードになる可能性は十分にあります。

これは単に一つの例にすぎませんが、このように、ごちゃごちゃしたロジックがリレー回路で組んであって、それらを全部正しく図面から読み取って、それぞれに対策をたてるのは、おそらく不可能に近いということが、図面を一見してわかりました。

そこで、資源的に見たときはもったいないことになるかもしれませんが、すべてのリレーロジック回路を切り捨ててしまっても、印字ならば印字する機械エレメントのぎりぎりのところまで、インターフェース回路の信号を送り込むことにしました。とにかく、エレメントが動きさえすれば、多少のロジックをソフトで持つことくらい何のことはありません。

この方法が良いか悪いかわかりませんが、われわれの力ではリレー回路の図面を読み切ることができなかったので、やむを得ずその道をとることにしました。機械を裸にしてしまったのですから、メーカーが指定するタイミングも何もあったものではありません。あとは実験でパラメータを決めるしかありません。以下で、われわれの悪戦苦闘の過程の一端を紹介します。

3.2 印刷装置のインターフェース回路

まず、このタイプライタは-50ボルトの電源をもっていて、これでリレーが動作するようになっています。リレーのコイルの一端は-50ボルトに接続されているので、他の一端を外に引っ張り出してきました(図3.1)。この引

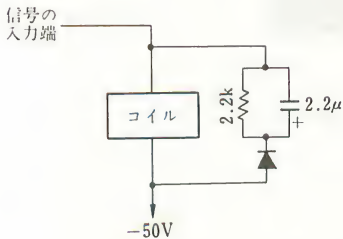


図3.1 コイルと消弧回路

き出してきた点をアースに落とせば、電流がコイルに流れ込んで内部のリレーの接点がメークされます。勿論、データライタのGNDとマイクロコンピュータのGNDは共通にしておきます。

研究室に古いパワートランジスタが沢山ありましたので、それで図3.2のような回路を作って実験したところ、最初のうちはおかしい動作をしましたが、ベース抵抗を取り換えて電流を調整したところ、正しく動作するようになりました。いまから考えてみると、ここにはもう一つトランジスタを入れてダ

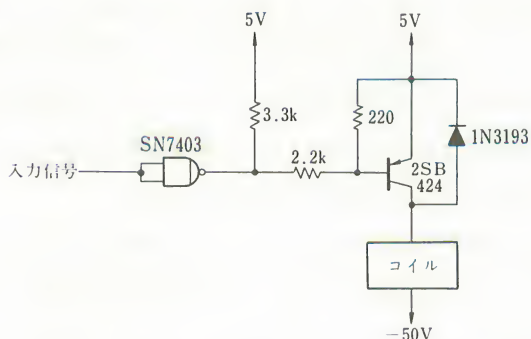
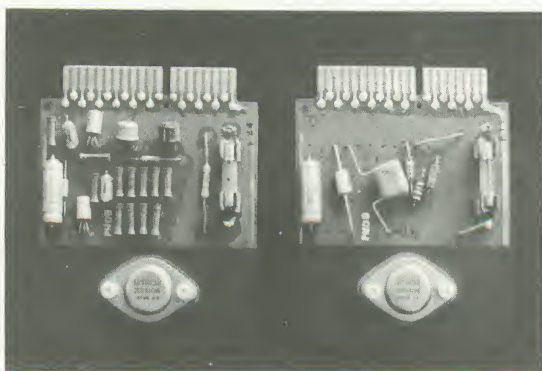


図 3.2 リレーのコイル駆動回路



(左)

(日立製作所の HIPAC から取り出してきたドライバ回路)

(右)

(パワートランジスタだけ残して、その他はこちらの設計部品をのせた回路)

写真 3.2 リレードライバ回路

ーリントン接続にしておけばよかったと思いますが、いずれにしてもリレーが正確に動作することには変わりありませんので、インターフェース回路からデータライタのリレーを動かすときには、すべてこの回路を使うことにしました。図 3.2 の回路をリレードライバ回路と名づけました(写真 3.2)。

次に、データライタを開けて印字のメカニズムを調べました。動力源として、一つの電動機が常時回転していて、これによって印字ハンマーのエネルギーが供給されます。

活字のどれを印字するかを選択(コードの設定)は 8 個のリレーによって

行なわれ、このリレーを適当にオンオフすると、それによって選択された活字が印字されます。このリレーの端子に $TSM_0 \sim TSM_7$ の記号をつけた。

印字を開始させるには、もう一つのリレーがあって、これに電流を流すと

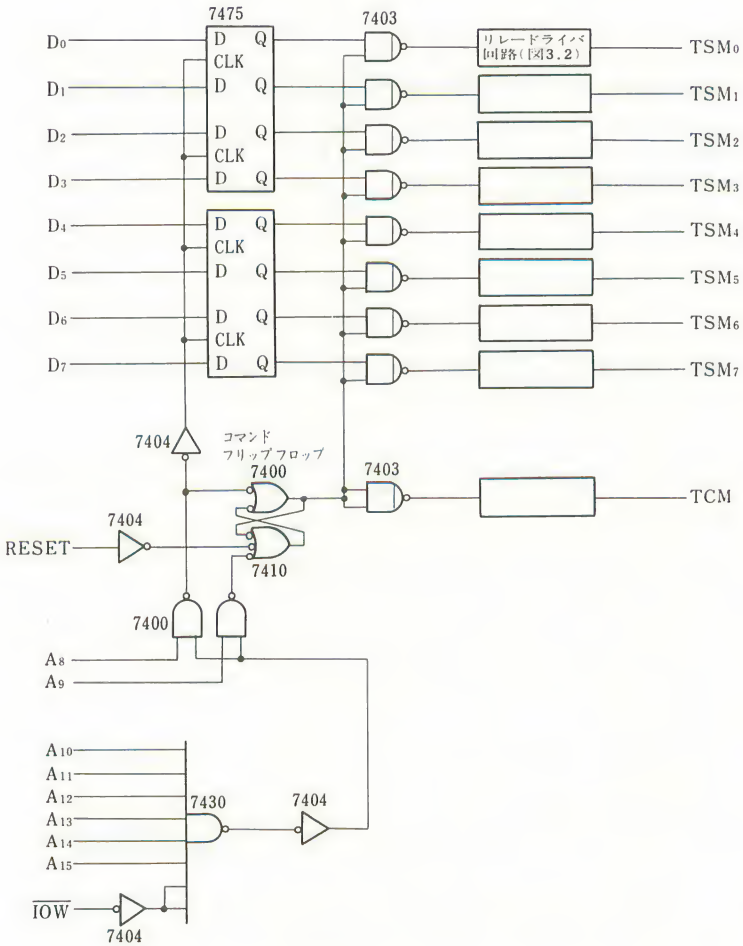


図 3.3 印字のタイミングを測定するための実験回路

78 第3章 入出力タイプライタのインターフェース

ラッチがはずれ、電動機の回転に引っぱられて印字動作が始まります。この端子にタイプコマンドという意味から TCM という名前をつけました。TCM に電流を流すと、活字の駆動カムがまわりはじめますが、それからしばらくの間に $TSM_0 \sim TSM_7$ のリレーにデータをセットすると、その活字が印字されることになります。

印字コマンド信号 TCM にどのくらいの時間にわたって信号を入れたらよいかわからなかったので、図 3.3 の実験回路を作って、マイクロコンピュータによる印字テストをしました。この回路で、まず、Aレジスタに印字したい活字のコードを置き、

OUT /FD

とすると、Aレジスタのコードが2つの7475にセットされ、同時に、コマンドフリップフロップがセットされ、TCM がオンになります。これでデータライタに印字開始の指令が入ります。そこで、頃あいをみはからって

OUT /FE

とすると、コマンドフリップフロップがオフになって、印字指令が解除されます。

この間に、時間をいろいろ変化させて実験を行ないました。もし、待ち時間が短かければ印字は行なわれず、逆に長すぎれば、2文字以上の印字が行なわれるはずです。こういう予想のもとに実際のデータをとったところ次の数値を得ました。

最小時間 14.4 msec

最大時間 109.1 msec

すなわち、印字コマンド TCM を 14.4 ミリ秒以上にすれば 1 文字の印字ができるが、109.1 ミリ秒以上にすると、2 文字つづけて印字してしまうということです。14.4 ～ 109.1 ミリ秒という時間幅があまりに大きいのでびっくりしました。パルス幅としてこのなかのどこをとったらよいかさっぱりわかり

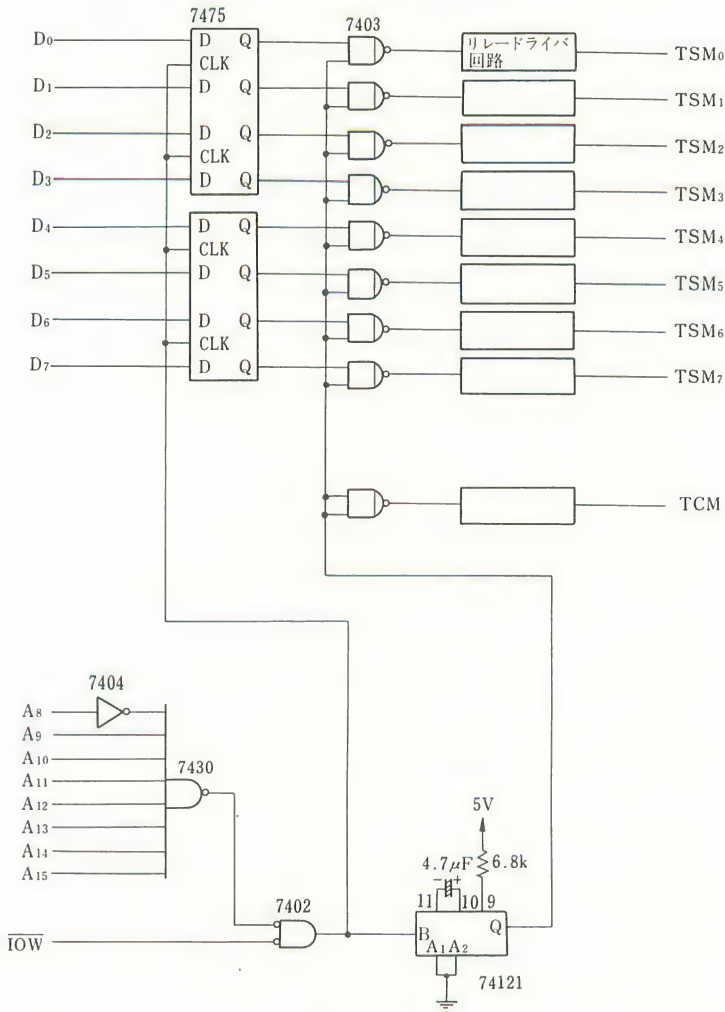


図 3.4 パルス幅を固定した印字回路

ません。電力を節約するという点からいえば、この時間は短いほうがよいのですが、信頼性という点からは、あるていど余裕を見込んでおいたほうが良いと思われます。

いずれにしても、一応の時間幅が得られましたので、図3.3のコマンドフリップフロップを図3.4のワンショット回路に置き換えて再実験しました。この場合、コンデンサが $4.7\mu\text{F}$ で抵抗が $(2 + 6.8) \times 10^3$ オームですから、時定数は

$$4.7 \times 10^{-6} \times 8.8 \times 10^3 = 41.36 \times 10^{-3} \text{ 秒}$$

となります。これに0.693をかけると大略のパルス幅が出ますから、

$$41.36 \times 0.693 = 28.16 \text{ msec}$$

となって、約30msecぐらいの印字コマンドができるはずです。

ここで、1文字ピタッと打てば万才ですが、実際はそうはいきません。マイクロコンピュータから

OUT /FE

とすると、2文字も3文字も続けて印字してしまいます。前の回路と比較すればわかるように、フリップフロップをワンショットに置き換えたのだから、原因はこのワンショットにあるにちがいありません。

そこで、ワンショットの V_{cc} からGNDにかけて $100\mu\text{F}$ のコンデンサを入れると、これでかなり落ちつきました。普通の場合ならば、ほとんど誤動作しません。さらに、AC電源にフィルタを入れると、ノイズテストをしてもOKとなりました。

こういった経験を通じて、ワンショットがあまり安定な回路でないことがわかりました。要するに、ワンショットは本質的にリニヤ回路の一種であって、ディジタル回路のように、ノイズマージンが明確になっていないので、これをディジタル回路に混用すると、回路全体の品質が低下する結果になります。これから、ワンショットの使用はできるかぎり避けることにしました。

しかし、そうはいつでも、図3.4の回路は私の研究室でかれこれ3年ぐらい連続的に使用していますが、これという事故もなく、無事に役目を果たしています。ときに誤動作が起こるようなことがあっても、「変だな」とブ

ツブツいいながら、もう一度やり直すと、たいいていの場合、正しい結果が得られます。とくに、「作り直せ」などというクレームは起こりません。こういうところが、大学の研究室の気楽なところです。

最後に、印字動作の終了をどうして知るかという問題が残っています。そこで、またまたデータライタの蓋をあけてなかをのぞきましたところ、印字動作の終了は3種の信号源から接点出力として取り出せることがわかりました。それらを簡条書きにすると、

- (1) 活字を印字したときの動作の終了信号
- (2) キャリッジリターンまたはタブ動作をしたときの動作の終了信号
- (3) 紙送り（ラインフィード）をしたときの動作の終了信号

の3種類です。これらの信号は常時-50ボルトになっていて、意味のある信号を出力するときに0ボルトになります。接点出力ですから、チャタリングも多いとみなければなりません。

そこで、図3.5のように、-50ボルトから0ボルトに振れる入力を0ボルトから5ボルトに振れる出力に変換する回路を作りました。これを電圧変換回路と呼びます。接点のチャタリングを吸収するために入力端には $1\mu\text{F}$ のコンデンサをつけ、高周波分をGNDに逃がします。その後は、-50ボルトのときにトランジスタをカットオフにし、0ボルトのときにオンにする抵抗をつければよいから、図のように 47k と 5.6k の抵抗で電圧を分圧します。A点が-50ボルトになったときのC点の電位を計算すると、

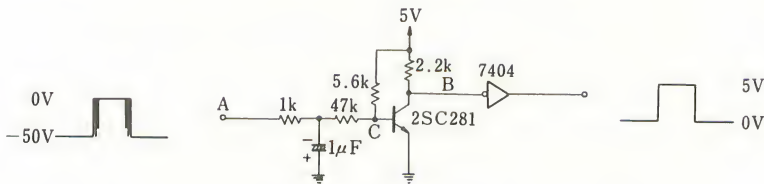


図 3.5 電圧変換回路

$$V_c = -50 + (5 + 50) \frac{47 + 1}{47 + 1 + 5.6} = -0.75$$

となりますから、トランジスタは明らかにカットオフになります。A点を0ボルトにすれば、C点の電位が正電位になることは明らかですから、トランジスタはオンになりますが、このとき、ベース電流を十分に流して、B点の電位が0ボルトに近くなるようにします。こうしてできたのが図3.5の電圧変換回路です。写真3.3は製作した電圧変換回路を示します。

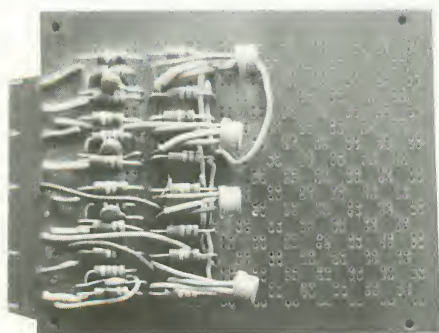


写真 3.3 電圧変換回路

印字動作の終了を知らせる信号源が3つありますので、図3.6のように、電圧変換回路を3個使用して、コレクタ直結によるワイヤードOR回路を作りました。印字、キャリッジリターン、TAB、ラインフィードの動作は排他的に起こるので、このような使用法が可能になります。

印字ステータスフリップフロップとして前に買っておいたシグネティック社DCL8224を使用しました。これは、クロックの立下りでセット・リセットされるACタイプのRSフリップフロップで、直接リセットRD端子もついています。

まず、CPUから負極性のパルスを送り、印字ステータスフリップフロップをリセットしておきます。次に、印字の開始信号TCMを出してタイプ動作をスタートさせますが、これは印字かキャリッジリターンかTABがラインフィードのいずれかです。動作がこれらのどれであっても、動作が終了すると、リレーの接点がメークされて出力端はGNDに上がり、それに対応するトランジスタのコレクタはロウに落ちて電流を吸い込みます。その後、

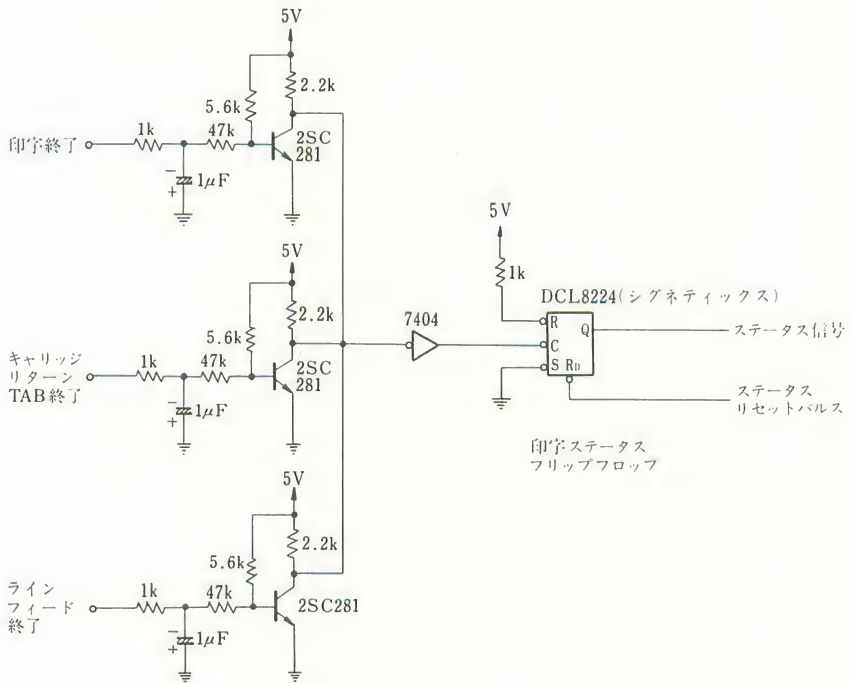


図 3.6 印字ステータスフリップフロップ

リレーが GND から落ちると、コレクタはハイに上がりますが、このときに印字ステータスフリップフロップにクロックが入り、ステータスを 1 にセットします。これでタイプ動作が終了しました。

実際に、種々の実験を行なってみました。キャリッジリターンの終了信号は、少し早目に出ることがわかりました。これは、おそらく、リミットスイッチの設置場所が悪いのではないかと考えましたが、メカニズムに手を入れるのはやっ介なのでそのままにしておきました。

キャリッジリターンをした直後に印字すると、活字の位置が少し上の所に来てしまって、文字が横一列に並ばなくなります。これはソフトウェアで補正することにしました。

さて、このようなインターフェース回路を採用すると、1文字印刷するときの手順は図3.7のようになります。

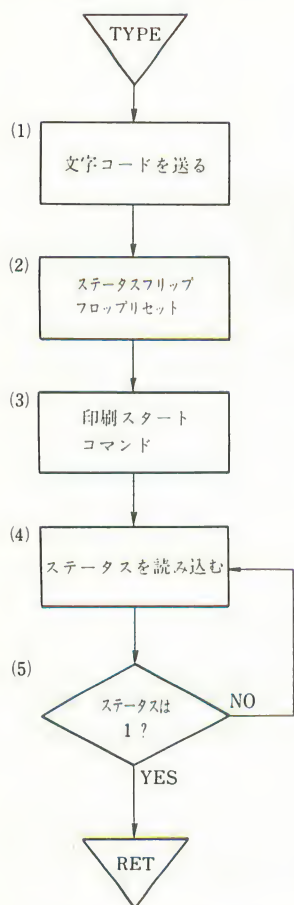


図 3.7 印刷サブルーチンのフローチャート

(1) 文字コードをインターフェース回路内のレジスタに出力する。

(2) ステータスフリップフロップをリセットする。

(3) 印刷開始コマンドを送る。

(4) ステータスを読み込む。

(5) 0 ならば動作未完了だから(4)に戻る。

1 ならば動作完了で主ルーチンに戻る。

これで何のさしさわりもあるわけではありませんが、経済性という点から、もっと良い方法があるかもしれません。この点について、もう少し詳しく検討してみます。

まず、印刷開始コマンドを送り、ステータスを読み込み、動作が完了するまで待ってリターンするところです。ここは機械が確実に動作を終了するのと確認してメインに戻るということですが、その間にマイクロコンピュータが実質的に何もしないのはもったいないではないか、という反対があるかもしれません。

そこで、このようにしたらどうでしょうか。

マイクロコンピュータは印刷開始コマンド TCM を発したら、サブルーチンからただちにメインルーチンに戻るようにします。そこで、必要な計算なり処理を行ない、その次に印刷する要求が発生した時点で、ステータスを見て

前の印刷動作が完了しているかどうかを調べ、もし完了していないときは、ここに待ちを入れるようにします。この場合のフローチャートは図 3.8 のようになります。図 3.8 の手順は図 3.7 の前半と後半を入れ換えたものになっています。

どちらがより良いかを実験的に調べるために、2つのプログラムを作っ

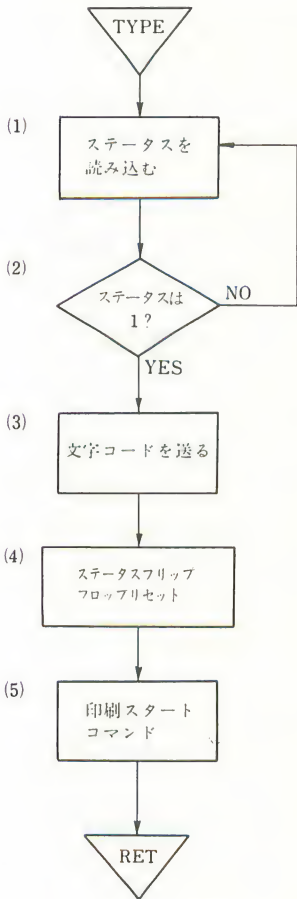


図 3.8 改良した印刷サブルーチン

いろいろ使ってみました。計算時間の点からいえばほとんど同じようなものでした。

要するに、タイプライタに印字するとき、いくつかの文字を続けて印刷する場合が多いので、図 3.7 を図 3.8 のように改良しても、その利点があり有効に使えないということがわかったのです。そういことならば、図 3.7 の手順のほうが、そのプログラムのなかで、すべての動作が完結しているという意味で優れていますから、現在、私の研究室では図 3.7 のアルゴリズムを採用しています。

動作が一つのプログラムのなかで完結しているという概念は非常に重要な概念であって、信頼性の高いシステムを設計するときに必ず採用されます。たとえば、マイクロコンピュータが印刷以外の動作をしているときに、ノイズで図 3.6 のステータスフリップフロップが 0 から 1 に反転したと仮定したとき、図 3.7 のアルゴリズムを採用している系ならば何の支障もなく次の印刷

が実行できるのに、図 3.8 のアルゴリズムを採用すると最初のループから永遠に抜け出せないという結果になります。

こういったところに、システムエンジニアリングの考え方が応用できるのであって、最初にもいったように、インターフェース回路の設計を小さく見れば、それは単にディジタル回路の作り方になりますが、アプリケーションも含めて考えると、これは完全にシステム工学の応用分野の一つになるのです。

印刷動作終了信号を割込み信号に応用することもできますが、これは読者の演習問題とします。

3.3 紙テープパンチ装置のインターフェース回路

以上で文字の印刷はほぼ完全にできるようになりましたので、次に、同じような性質をもっている紙テープパンチのインターフェース回路を作ることになりました。パンチ部のメカニズムはデータライタに向かって左側について、それだけ独立に取りはずしできるようになっています（写真 3.4）。

機械の中味を調べたところ、装置のメカニズムは印刷の場合とかなり違っていましたが、コントロールの取り方は印刷の場合とほぼ同じ原理で作られていることがわかりました。

まず、8 個のデータをセットするリレーがあります。これに $PSM_0 \sim PSM_7$ の信号名をつけました（印字の場合は $TSM_0 \sim TSM_7$ だった）。これに電流を流すと、紙テープの対応する場所に穴があけられ、電流を流さないと穴はあけられません。個々のリレーと対応する穴の位置は装置のなかの配線を目で追ってもわかりますが、実際にコードを打たせることになっても知ることができます。このほうが簡単です。

紙テープパンチの動作開始指令を出すリレーがあり、これにパンチコマン



写真 3.4 紙テープパンチ部

ドPCMという名前をつけました(印字の場合はTCM)。PCMの端子をGNDに落とすと、コイルに電流が流れ、ストッパーがはずれてパンチ機構駆動軸が回転をはじめます。回転開始の最初の時期に $TSM_0 \sim TSM_7$ にセットしておいたオンオフ情報が機械的に記憶され、そのパターンが紙テープの上に穴となってあけられます。

パンチのサイクルの最後のところで、紙テープが1文字だけ送り出され、次のパンチが可能な状態になります。これで1文字(8ビット)のパンチ動作が完全

に終了しました。

このように、文字の印字動作と紙テープのパンチ動作は完全に1対1の対応がつくように作られているので、図3.3の実験回路をそのまま使ってタイミングを測定することにしました。図において、 $TSM_0 \sim TSM_7$ を $PSM_0 \sim PSM_7$ に変更し、TCMをPCMに変更することだけで紙テープパンチができました。

コマンドフリップフロップをセットしておく時間の最大値と最小値を測定したところ、次のような実測値を得ました。

最小時間 7.3 msec

最大時間 45.8 msec

すなわち、PCMに電流を流す時間は最小7.3 msecから最大45.8 msecまで幅があります。7.3 msec以下だとパンチせず、45.8 msec以上にすると2回続けて同じパターンをパンチしてしまいます。

この時間幅は文字を印字する場合の時間幅よりもかなり小さくなっています。これは、おそらく印字とパンチの機械のメカニズムの違いからくるものと思われますが、インターフェース回路を設計する立場からいえば、これだけのデータで十分ですから、それ以上せんさくするのはやめておきました。

ということで、紙テープパンチのインターフェース回路は図3.4とまったく同じに設計しました。したがってワンショット回路のパルス幅も前と同じで約30msecとなります。今回の実験データからいうと

$$7.3 < 30 < 45.8$$

となっていますから差しさわりのないのですが、もう少し小さい値のほうが良かったと思います。しかし、同じ回路板がどちらにも使えるということにも、魅力があるので、このようにしました。

紙テープパンチ動作の完了は一つのリミットスイッチから送られてきます。これにPCCという名前をつけました。パンチステータスフリップフロップは図3.9のように設計しました。考え方の基本は図3.6の場合とまったく同じです。

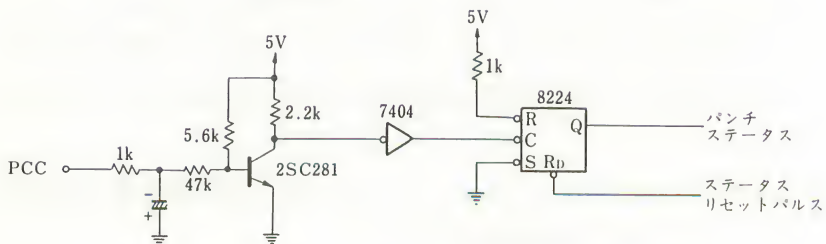


図3.9 パンチステータスフリップフロップの周辺回路

1バイトのデータを紙テープの上にバイナリー形式でパンチする手順は図3.10のフローチャートになります。図3.7と図3.10を比較すればわかるように、文字の印字と紙テープパンチという機械的にまったく違った動作が、インターフェース回路の作り方次第で、ソフトウェア的構造がまったく同じに

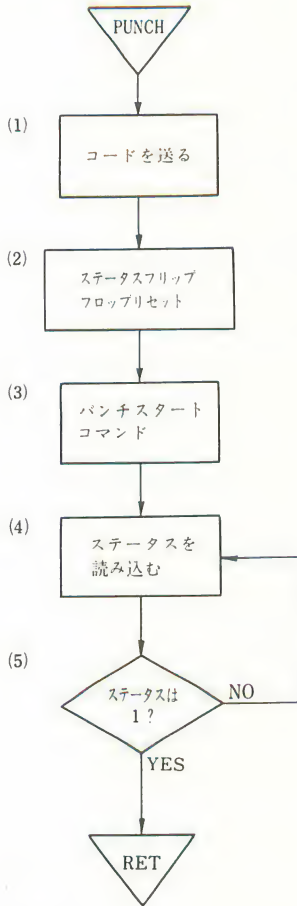


図 3.10 パンチサブルーチンのフローチャート

なってしまふことがわかります。

もし、文字の印刷と紙テープパンチを同時に行なわないとすれば、7475のDラッチも4個は必要でなく、2個を共用すればよいことがわかります。ミニコンピュータのオペレーティングシステムに慣れた人から見れば、せっかく並列処理ができるところをそれをできなくするのはもったいないようにみえるかもしれませんが、マイクロコンピュータの場合は並列処理のためのオーバーヘッドをあまり大きくとることはできないので、直列処理にしたほうが良い場合が多くあります。こういうところは、じっくり損得を考えたうえで判断を下さなければなりません。

3.4 紙テープ読み込み装置のインターフェース回路

次に、紙テープにパンチされている情報をCPUに読み込む装置を作ることになりました。この装置は写真3.5のように、データライタに向かって右側に、自由に取りはずしできるような形でついています。

この場合は、前2例と違って、装置からの情報がCPUに読み込まれます。情報の流れの向きが逆になっていることに注意して下さい。

装置のメカニズムを調べた結果、次のような信号がこの動作に関係していることがわかりました。

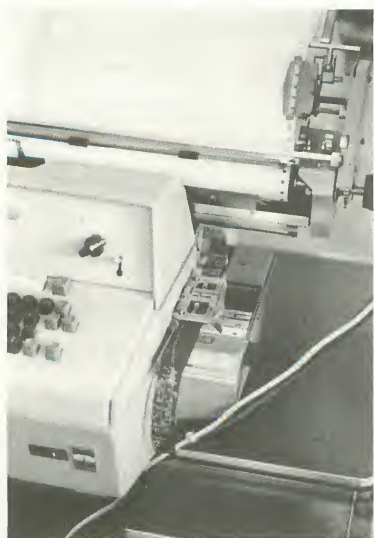


写真 3.5 紙テープリーダ部

まず、 $RC_0 \sim RC_7$ の受けの回路ですが、これは本質的には図3.5の電圧変換回路で良いのですが、接点がメークしたときに0ボルト、ブレイクすると入力端が浮いてしまいますので、図3.11のように、入力端を6.8k Ω の抵抗で-50ボルトにプルダウンしました。

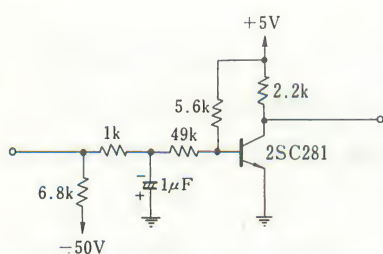


図 3.11 紙テープリーダのための電圧変換回路

(1) リードコマンド RCM ; CPU が装置に与える信号で、リーダのストッパーをはずし、紙テープ上の情報を1文字分だけ読み込む動作を開始する。

(2) $RC_0 \sim RC_7$; 装置が紙テープを読み取った結果が出力される。接点出力で、紙テープ上に穴のあいているとき0ボルト、そうでないとき接点はブレイクされている。

(3) RCC ; 装置がインターフェースに出力する信号で、読取り動作中のときに0ボルト、そうでないときに-50ボルトとなる。

次にリードコマンド RCMのパルス幅の決定ですが、これは前2例と同様に実験的に決めました。そのプロセスを再述するのは省略して結果だけを示すと図3.12のようになります。パルス幅を計算すると、

$$0.693 \times (33 + 2) \times 10^3 \times 1 \times 10^{-6} = 24.26 \times 10^{-3}$$

となって、約25 msecのパルスを出していることになります。前2例に比べ

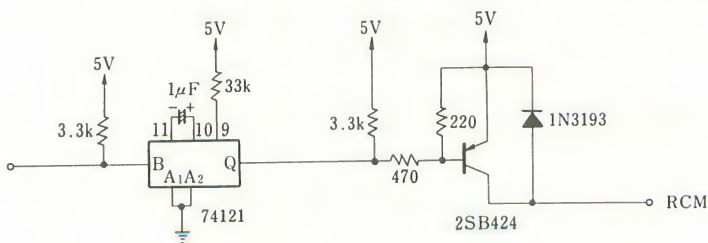


図 3.12 リードコマンド発生回路

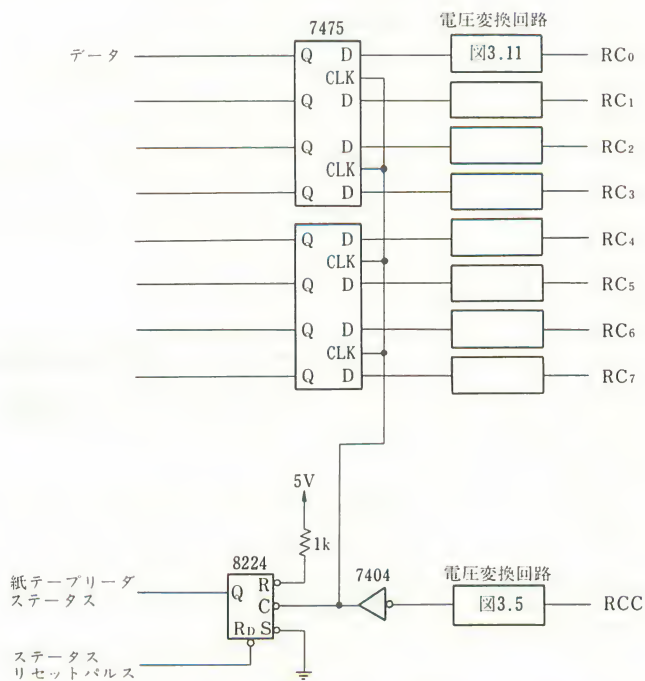


図 3.13 紙テープリーダーのインターフェース回路

て約 2 割細くしました。

最後に、読み出されたデータをインターフェースのフリップフロップにセ

ットするタイミングと動作完了のステータスを作るところの回路ですが、これらはすべて RCC 3 の後縁を使えばよいことがわかりました。RCC の後縁は動作の完了を示すタイミングですから、ステータスのセットに使うことは当然です。読み取ったデータは実はこのタイミングよりもかなり前に出ているのですが、その信号源が見つからなかったためこのように設計しました。

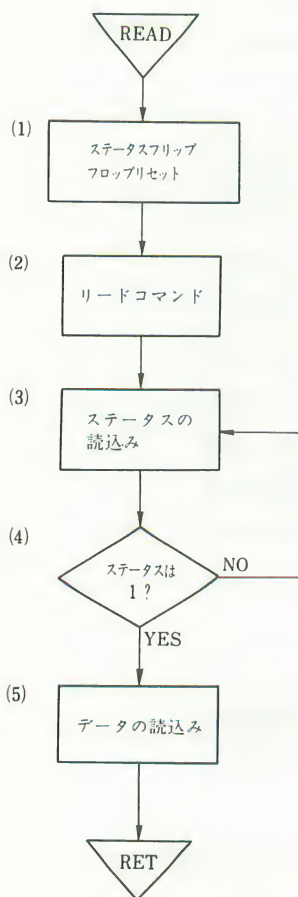


図 3.14 紙テープ読み込み手順のフローチャート

一度読み出されたデータは、次のデータが読み出されるまで機械的に保持されていますから、このような処理が可能になります。実際の回路は図3.13のとおりです。

以上のインターフェース回路を使って、紙テープリーダーから1文字の情報をCPUに読み込む手順をフローチャートに書くと図3.14のようになります。これは、とくに説明する必要はないでしょう。

3.5 キーボード読み込み装置のインターフェース回路

最後に、キーボードからデータを読み込む回路について考えます。最初に説明したように、今回の入出力タイプライタはオフライン用をオンライン用に改造したものです。このために、オフライン機器としての癖がメカニズムのなかに残っていて、どうしても取り除くことができない所がいくつかありました。

キーボードからデータを読み込むところがそ

の一つであって、この装置ではキーボードのキーを押すと、必ず押した文字が印字されます。このところは機械が自動的に行なうので、切り離しはできませんでした。

さらに印字されたデータのコードは $RC_0 \sim RC_7$ の信号線に出てきますので、紙テープを読んでもキーボードを押しても、同じ信号線に出てくこと

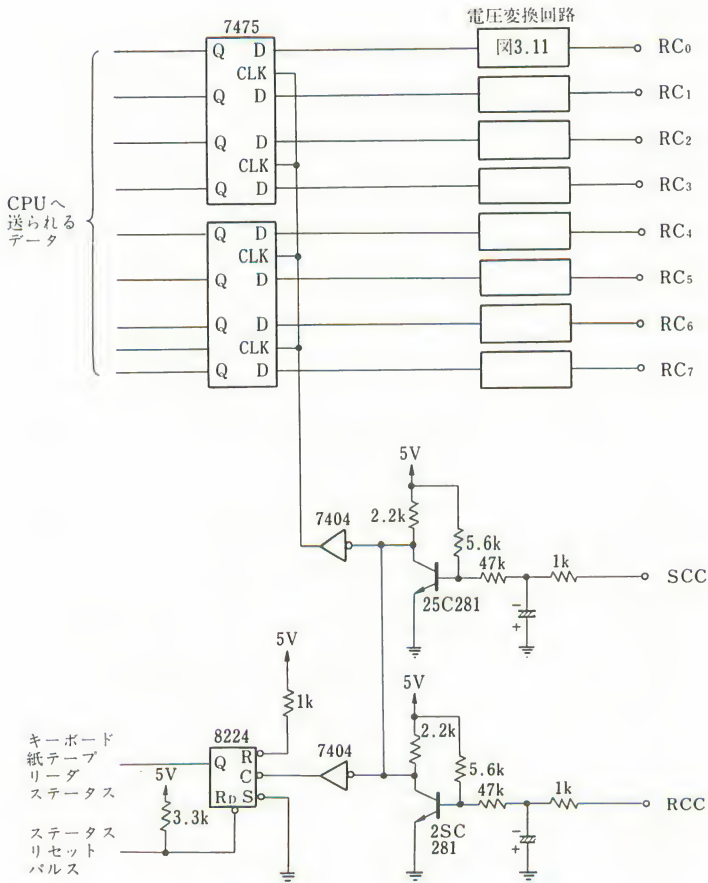


図 3.15 キーボードリードと紙テープリードを統合したインターフェース回路

になります。紙テープ読取りのときは、RCCに動作完了信号が出てきましたが、キーボードの場合はSCCという信号線に完了が出力されます。

紙テープリーダーのときは、外部からRCMのパルスを入れないとリーダーの動作がスタートしませんが、キーボードの場合は、人間が手でキーボードを押すことがスタート信号になっていますから、CPUからのコマンド信号を入れる必要はありません。

キーボードのコードと紙テープ読取り信号がまったく同じリレーから出てくるので、これを利用して図3.15のように、同じフリップフロップにデータ

をセットすることにした。ワイヤードOR回路です。したがって、紙テープ読込み動作中にキーボードを押したりすると、誤動作の原因になります。

キーボードから1文字を読み取るサブルーチンのフローチャートは、図3.16です。この流れ図からもわかるようにサブルーチンに飛んできて、最初にステータスフリップフロップをリセットするので、それ以前にもしキーが押されていても、それはクリアされてしまってCPUには読み込まれないということになります。このサブルーチンに飛んできた後に押されたキーの内容がプロセッサに取り込まれます。

なぜこのようなことをしつつこくいうかといいますと、インターフェース設計という面からみると紙テープ読取りもキーボード読込みもまったく同じに見えるのですが、もっと高い立場、すなわちシステムエンジニアリングの立場から

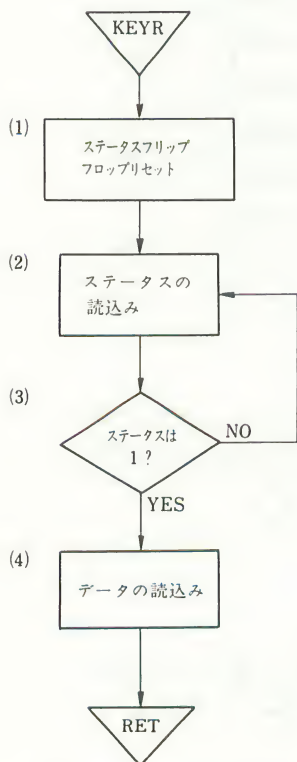


図 3.16 キーボードから1文字読み込むサブルーチン

見ると、紙テープ読取り動作は完全にCPUのコントロールのもとに進行するのに、キーボード読込みの場合はキーボードを使う人間の意志がコンピュータとはまったく独立に働くという点で、性質のまるで違う対応が必要になってくるからなのです。

このような種類の問題をマンマシンインターフェースの問題といいます。ここでは、この問題にはあまり深入りしないで、そういう問題があるということを指摘するだけにとどめておきます。とにかく、マイクロコンピュータと人間が接触するところでは、設計を慎重にしなければならないことは事実です。人間は何をするか、まったく予想できないという一面をもっています。機械が破損するというような事故も、多くこのようなところから発生するものです。

最後に、キーボードをロックするリレーがありましたので、これは図3.17の回路を作り、CPUからキーボードをロックしたり、またはそれを解除したりすることができるようにしました。

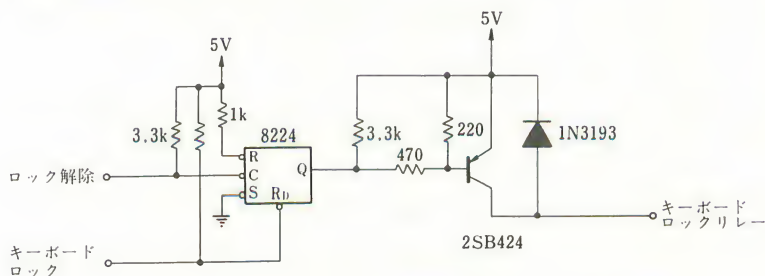


図 3.17 キーボードロックと解除フリップフロップ

3.6 低速バスのインターフェース回路

以上述べてきたように、実験的に確かめながら改造したデータライタを8080 Aのマイクロコンピュータシステムに接続することにします。まず、

96 第3章 入出力タイプライタのインターフェース

8080 AのCPUカードですが、これは前章の図2. 16で説明したものと、バス構成にかんするかぎり同じです。私の研究室で実際に使用しているシステムではフロントパネルのコントロールカードが1枚入っていますが、これについては、今回ふれないことにします。パネルインターフェースカードについて、とくに興味のある読者は「マイクロコンピュータの作り方」(産報出版)を参考にして下さい。

コネクタは100極のものを使用しました。最初に、マザーボードを10スロットルで製作しましたが、このへんのところの見通しが多少甘かったようで、メモリを増加していくにしたがって、スロットルの数が不足してきて、動きがとれなくなりました。現在、システムのなかに入っている基板を列記すると次のようになります。

- (1) CPUカード8080 A 1枚
- (2) フロントパネルコントロールカード 1枚
- (3) RAMカード, 2102 A-4, 5枚 $4\text{ KB} \times 5 = 20\text{ KB}$
- (4) PROMカード, 2708 16個分, 現在書き込み済みのものが3 KBで,
その他はソケット 1枚
- (5) 画像入力インターフェースカード, これについては別の機会にあらためて述べます, 1枚

これで基板が全部で9枚になり、残りは1枚しかありません。これをデータライタのインターフェース回路に当てると、それ以後一つの入出力機器も接続できなくなります。

私たちのような大学の研究室では、研究のテーマごとにプログラムが違いますから、どうしてもRAMの容量が大きくなります。というのは、テーマのなかで一番大きいプログラムにマイクロコンピュータのRAMの容量が右へならえをしてしまうからです。常時20 KバイトのRAMをフルに使っているということではありません。

しかし、いずれにしても、システムのバスに、これ以上の基板が入らないことがはっきりしましたので、応答の速さを必要としない入出力機器のインターフェース回路は別のラックのなかに入れることにしました。これを以下で低速バスと呼びます。

スロットルの数に余裕があれば、データライタのインターフェース回路を1枚の基板の上に作りあげるのが良いにきまっています。現在、この低速バスにはXYプロッタ、CRTディスプレイ、高速紙テープリーダー、高速ラインプリンタ、カセットMTなどの装置が接続されていますが、いずれも順調に動作しています。

バス増設のための低速バスを設計するに当たって、最初に問題になるのはケーブルが長くなったときの対策です。一般に、スタンダードTTLの出力端から、次の入力端へのケーブルの長さは20～30cm以下とされています。逆にいえば、それ以上の長いケーブルを引くときには、別途の対策を立てなければならぬということです。実際に、入出力装置へのケーブルの長さは、数メートルになりますから、特別な対策を立てることが必要になるのは当然です。

いくつかのミニコンピュータのバス構成などを調べましたが、使用するケーブルの特性インピーダンスも関係してくるので、そう簡単にパラメータを決定することができません。細かいところは実験で決めることにして、大筋をだいたい次のように決めました。

- (1) データはなるべくフリップフロップなどにラッチして、スタティクな信号を使用する。信号の立上りや立下りの近傍を避ければ、普通のTTLゲートでもかなり遠くまで信号を確実に送ることができる。
- (2) どうしてもダイナミックなパルスを送信しなければならないときには、**図3. 18の回路形式を統一的に採用する。**

ここで、送信側の素子としてSN7438を用います。これはオープンコレクタ

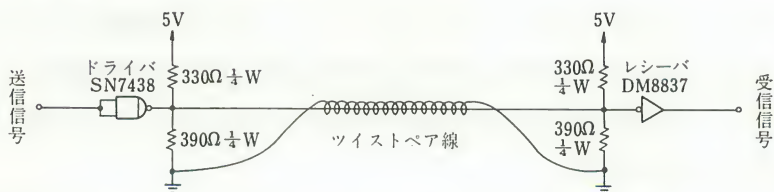


図3.18 ケーブルを長く引く場合のインターフェース回路

の2 NANDゲートですが、シンク電流がスタンダードTTLゲートに比較して約3倍の能力があります。これを390 オームと330 オームの抵抗で固定すると、ドライバがオフのときの電圧が

$$5 \times 13 / 24 = 2.71$$

となって約3ボルトぐらいになります。

伝送線はツイストペア (twisted pair line) を使用し、受信端でも同じ抵抗でケーブルを固定します。この抵抗値は使用するケーブルの特性インピーダンスと関係していますから、ケーブルによって適当な値に設定しなければなりません。

ケーブルの特性インピーダンスを測定するには、ブリッジが必要ですが、それが入手できないときは、シンクロスコープなどを使って、波形をみながら調整しなければなりません。しかし、この特性インピーダンスの値はかなり微妙で、ケーブルに人体が近づいたり、ツイストの巻き方を変えたりすると、インピーダンスも変化するので注意しなければなりません。

レシーバには1ボルトのヒステリシスをもつNS社のDM8837を使用しました。参考のために、図3.19にナショナルセミコンダクタ社の資料を再掲します。私の研究室では、このペアで現在5メートルぐらいのケーブルを引っぱっていますが、確実にパルスを送送することができます。われわれの経験からいえば、もっと長くケーブルを引いても大丈夫のようですが、その必要がないので、現在までのところ確かめてありません。さて、以上の準備をし

ておいて、低速バスの設計に入ります。

まず、データの出力ですが、これは3通りのタイプを作りました。第1は、基板のなかでフリップフロップにラッチして、完全にスタティックなデータにするものです。これを以下でモードⅠと呼びます。

図3.20の回路において、74154の4ライン→16ラインデコーダ回路によって、アドレスをデコードしてポート番号を作ります。一方で、データバスから信号をひろってきて、これを7475の入力に入れます。74154の15の端子からNOT回路を通して、クロックを作ると、データバスの内容がラッチされます。この出力信号に、一般に

$Q8X_x$

という信号名をつけます。たとえば、 $Q8F_3$ を1にセットしたいときには

```
MVI  A, 8
OUT  /8F
```

という命令を実行すればよいことになります。 $Q8F_6$ と $Q8F_0$ を同時に1にセットしたいときには

```
MVI  A, /41
OUT  /8F
```

とすればよいでしょう。 $Q8E_5$ を1にセットしたいときはどうすればよいですか。皆さんで考えて下さい。

第2は、基板のなかでフリップフロップにラッチしないで、データバスと書込みパルスゲートをゲートしてパルスを作り、これをダイナミックな情報として入出力装置に送るものです。これ以下で、モードⅡの出力と呼びます。図において、データバスの情報とストロブ信号を7438でゲートすると所期のパルスが得られます。これはダイナミックな情報なので、図3.18で述べた対策をほどこします。このパルスに、一般に

$\overline{P8X_x}$

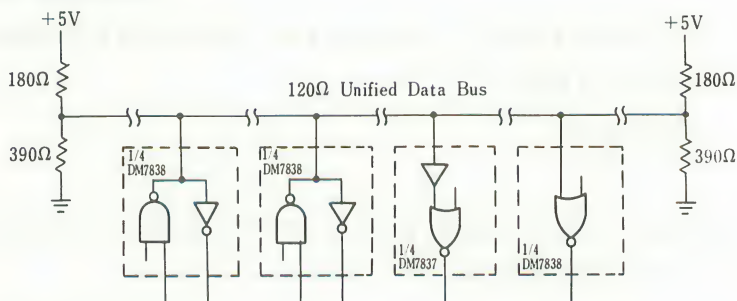
Series 54/74

DM7836/DM8836 quad NOR unified bus receiver

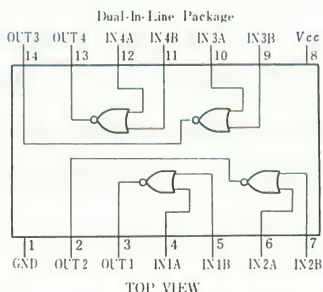
general description

The DM7836/DM8836 are quad 2-input receivers designed for use in bus organized data transmission systems interconnected by terminated 120 Ω impedance lines. The external termination is intended to be 180 Ω resistor from the bus to the +5 V logic supply together with a 390 Ω resistor from the bus to ground. The design employs a built-in input hysteresis providing substantial noise immunity. Low input current allows up to 27 driver/receiver pairs to utilize a common bus. The receiver has been specifically configured to replace the SP380 gate pin-for-pin to provide the distinct advantages of the DM7837 receiver design in existing systems.

typical application



connection diagram



features

- Plug-in replacement for SP380 gate
- Low input current with normal V_{cc} or $V_{cc} = 0$ V (15 μ A typ)
- Built-in input hysteresis (1V typ)
- High noise immunity (2V typ)
- Temperature-insensitive input the sholds track bus logic levels
- DTL/TTL compatible output
- Matched, optimized noise immunity for "1" and "0" levels
- High speed (18 ns typ)

absolute maximum ratings (Note 1)

Supply Voltage	7.0 V
Input Voltage	5.5 V
Power Dissipation	600mW
Operating temperature range :	
DM7836	-55 °C to +125 °C
DM8836	0 °C to +70 °C
Storage Temperature Range	-65 °C to +150 °C
Lead Temperature (Soldering, 10 sec)	300 °C

electrical characteristics

The following apply for $V_L \leq V_{CC} \leq V_H$, $T_L \leq T_A \leq T_H$, unless otherwise specified
(Note 2)

PARAMETER	INPUT	OUTPUT	COMMENTS	MIN	TYP	MAX	UNIT
High Level Input Threshold :							
DM7836	V_{TH}	16 mA	Output < 0.4 V	1.65	2.25	2.65	V
DM8836	V_{TH}	16 mA	Output < 0.4 V	1.80	2.25	2.50	V
Low Level Input Threshold :							
DM7836	V_{TH}	-400 μ A	Output > 2.4 V	0.97	1.30	1.63	V
DM8836	V_{TH}	-400 μ A	Output > 2.4 V	1.05	1.30	1.55	V
Maximum Input Current	4 V		$V_{CC} = V_H$		15	50	μ A
Maximum Input Current	4 V		$V_{CC} = 0$ V		1	50	μ A
Logic "1" Output Voltage	0.5 V	-400 μ A		2.4			V
Logic "0" Output Voltage	4 V	16 mA			0.25	0.4	V
Output Short Circuit Current	0.5 V	0 V	$V_{CC} = V_H$	-18		-55	mA
Power Supply Current	4 V		Per Package		25	40	mA
Input Clamp Diode Voltage	-12mA		$T_A = 25^\circ\text{C}$		-1	-1.5	V
The following apply for $V_{CC} = 5$ V, $T_A = 25^\circ\text{C}$ unless otherwise specified.							
Propagation Delays :							
Input to Logic "1" Output			Note 3		20	30	ns
Input Logic "0" Output			Note 4		18	30	ns

Note 1 : Voltage values are with respect to network ground terminal.

Positive current is defined as current into the reference pin.

Note 2 : For DM7836 : $V_L = 4.5$ V, $V_H = 5.5$ V, $T_L = -55^\circ\text{C}$, $T_H = +125^\circ\text{C}$.

For DM8836 : $V_L = 4.75$ V, $V_H = 5.25$ V, $T_L = 0^\circ\text{C}$, $T_H = +70^\circ\text{C}$.

Note 3 : Fan-out of 10 load, $C_{LOAD} = 15$ pF total, measured from

$V_{IN} = 1.3$ V to $V_{OUT} = 1.5$ V, $V_{IN} = 0$ V to 3 V pulse.

Note 4 : Fan-out of 10 load, $C_{LOAD} = 15$ pF total, measured from

$V_{IN} = 2.3$ V to $V_{OUT} = 1.5$ V, $V_{IN} = 0$ V to 3 V pulse.

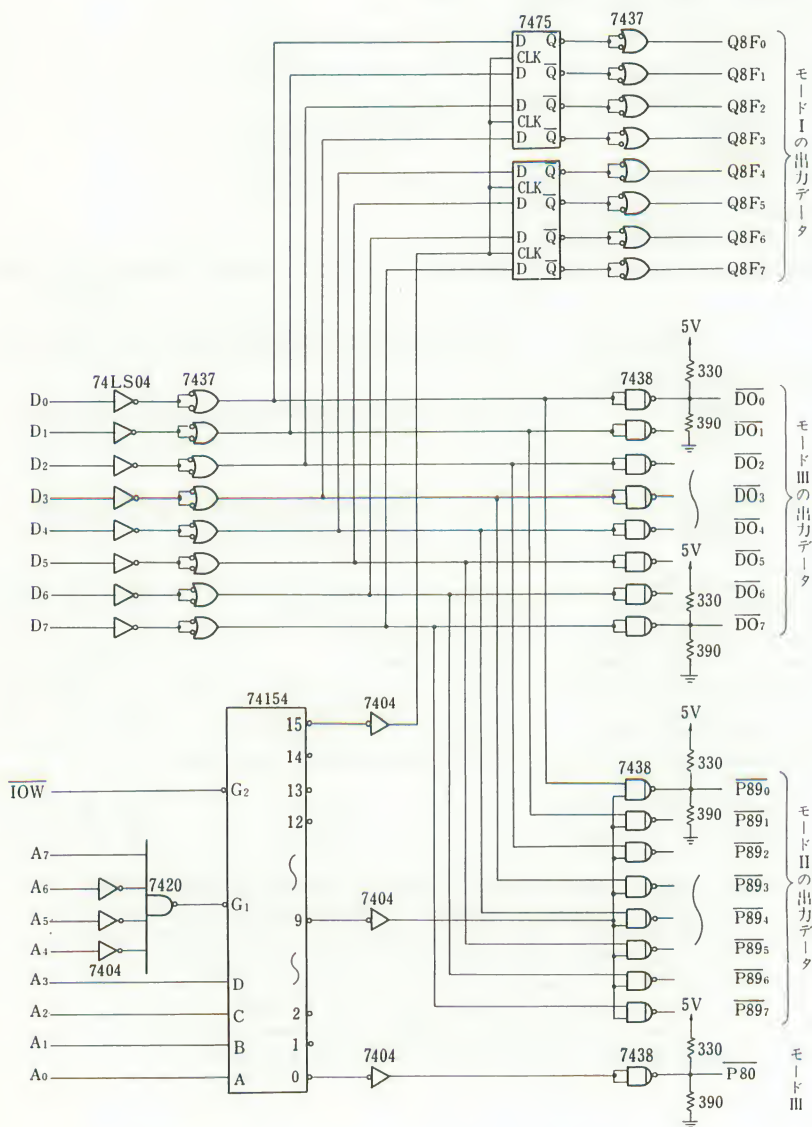


図 3. 20 低速バスの出力インターフェース回路

という信号名をつけます。たとえば、 $\overline{P89_4}$ に負極性のパルスを出すには、

```
MVI A, /10
```

```
OUT /89
```

とすればよいことがわかります。もし $\overline{P89_7}$ と $\overline{P89_1}$ に同時にパルスを出したいときには

```
MVI A, /82
```

```
OUT /89
```

とします。 $\overline{P88_5}$ にパルスを出すにはどうすればよいかわかりません。皆さんで考えて下さい。

第3は、データバスを外部に延長して、書込みパルスをダイナミックな形で装置に送り、そちらでどうとも自由にお使いなさい、というものです。これをモードⅢと呼びます。データバスも書込みパルスも、共にダイナミックな情報なので、図3. 18の対策をほどこします。とくに、データバスのほうは長く引っぱらないことに注意しなければなりません。

低速バスの入力としては、図3.21に示すオープンコレクタゲートによるワイヤードOR回路を採用しました。とくに、各ゲートの選択信号として、

```
Q8F0~Q8F7
```

のスタティック情報を使うことにしました。したがって、たとえば図3. 21のように $Q8F_2$ を使った入力ポートからデータを読み込む手順は、

```
MVI A, 4
```

```
OUT /8F
```

```
IN /80
```

となります。 $Q8F_2$ の代わりに、もし $Q8F_5$ を使ったとしたら読み込み手順はどうなりますか。皆さんで考えて下さい。

この章の次節と次の章で、ここで設計した低速バスを使いますので注意して下さい。

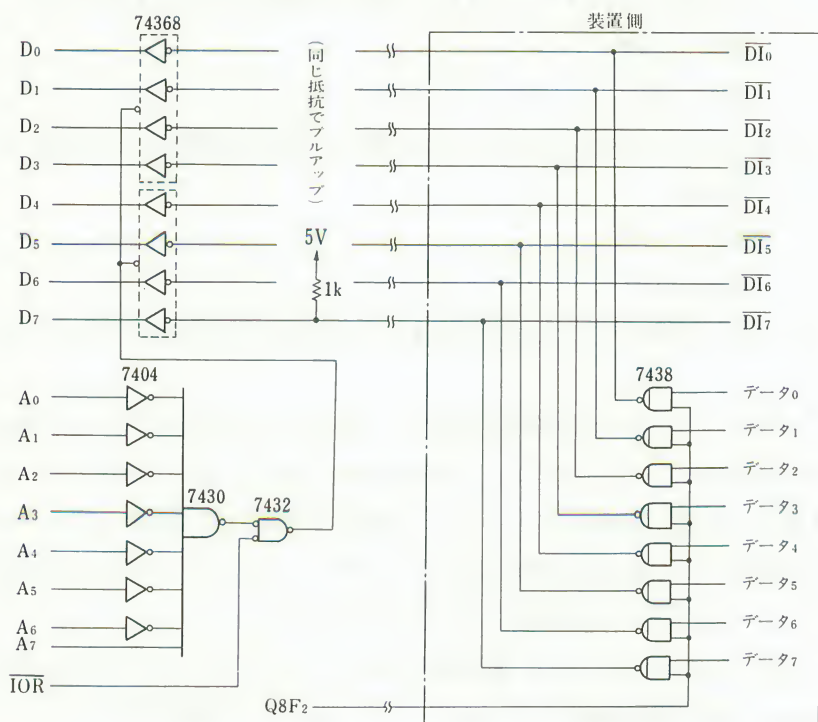


図3.21 低速バスの入力インターフェース回路

3.7 データライタのインターフェース回路と 基本サブルーチン

最後に、データライタの各駆動部と低速バスを接続した結果について述べます。

まず、印字回路ですが、これは図3.22のように配線しました。このとき、データライタに1文字印字するサブルーチンは次のようになります。➡

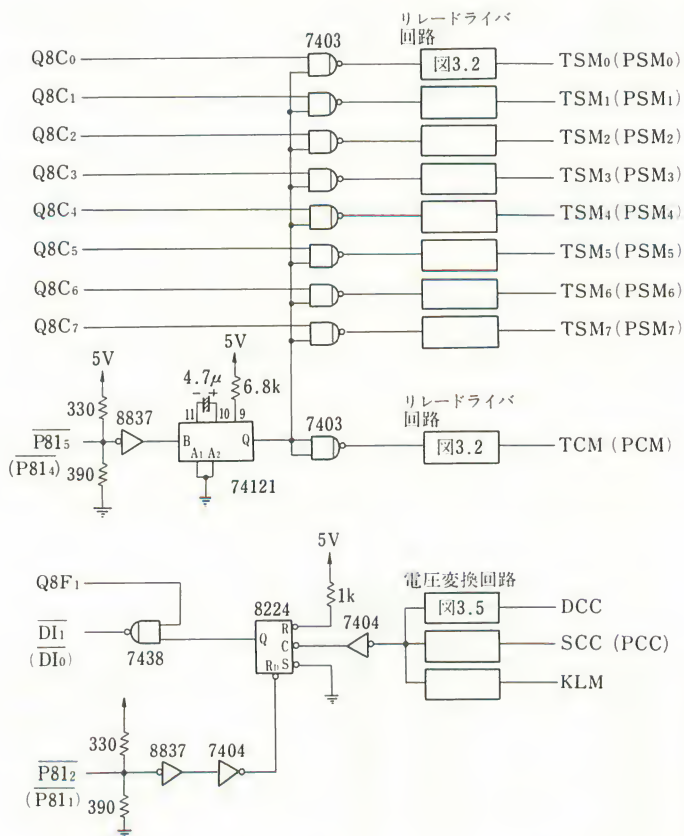


図 3.22 印字装置のインターフェース回路
() 内は紙テープパンチの場合

```

*TYPE SUBROUTINE
*TYPE ONE CHARACTER
*CODE IN A REGISTER

4058 D38C TYPE ORG /4058
405A 3E04 OUT /8C
405C D381 MVI A,4
405E 3E20 OUT /81
4060 D381 MVI A,/20
4062 3E02 OUT /81
4064 D38F MVI A,2
OUT /8F

*SEND CODE
*CLEAR STATUS
*TYPE COMMAND
*STATUS ADRS

```

4066	DB80	TYP1	IN	/80	*STATUS READ
4068	E602		ANI	2	
406A	CA6640		JZ	TYP1	*FLAG SET
406D	C9		RET		
			END		

約束として、Aレジスタに印字したい文字のコードをもってサブルーチンに飛んでくるものとしますが、サブルーチンから帰ったときAレジスタとフラグ類は保存されません。

次に、紙テープパンチ回路のインターフェースですが、これは本質的に印字の場合と同じです。図3.22の()内の配線のとおりにしました。ここで、印字と紙テープパンチを同時に別のコードで行なうことはまずありませんので、データのソースは同じにしました。Aレジスタに1文字分の情報をもったときに、それを紙テープにパンチするサブルーチンは、次のようになります。

			*PUNCH SUBROUTINE		
			*PUNCH ONE CHARACTER		
			*CODE IN A REGISTER		
			ORG	/408A	
408A	D38C	PUNC	OUT	/8C	*SEND DATA
408C	3E02		MVI	A,2	*CLEAR STATUS
408E	D381		OUT	/81	
4090	3E10		MVI	A,/10	*PUNCH COMMAND
4092	D381		OUT	/81	
4094	3E02		MVI	A,2	*STATUS ADRS
4096	D38F		OUT	/8F	
4098	DB80	PUN1	IN	/80	*STATUS READ
409A	E601		ANI	1	
409C	CA9340		JZ	PUN1	*FLAG SET
409F	C9		RET		
			END		

この場合も、Aレジスタの内容は保存されません。

次に、紙テープリーダとキーボード読み込み回路のインターフェースですが、これは図3.23のように配線しました。紙テープの情報を1文字分だけ読み取って、Aレジスタにセットするサブルーチンは次のようになります。

			*READ SUBROUTINE		
			*READ ONE CHARACTER FROM PTR		
			*CODE IN A REGISTER		
			ORG	/BE6	
0BE6	3E08	READ	MVI	A,R	*CLEAR STATUS
0BE8	D381		OUT	/81	

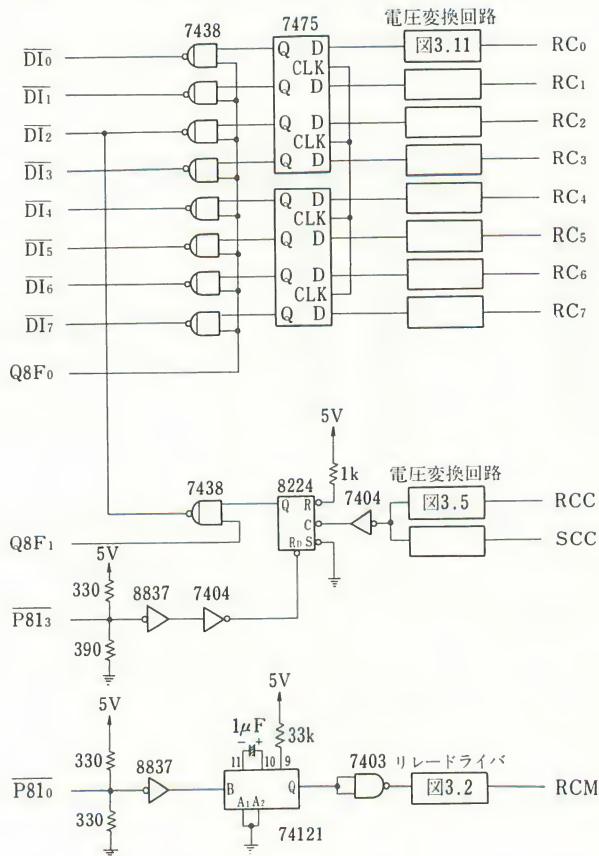


図 3.23 キーボードリーダと紙テープ読取り器のインターフェース回路

0BEA 3E01	MVI A,1	*READ COMMAND
0BEC D381	OUT /81	
0BEE 3E02	MVI A,2	*STATUS: ADRS
0BF0 D38F	OUT /8F	
0BF2 DB80	IN /80	*READ STATUS
0BF4 E604	ANI 4	
0BF6 CAF30B	JZ RED1	*READ COMPLETE
0BF9 3E01	MVI A,1	*DATA ADRS
0BF8 D38F	OUT /8F	
0BFD DB80	IN /80	*READ DATA
0BFF C9	RET	
	END	

Aレジスタ以外のレジスタの内容は保存されますが、フラグ類はこわれます。キーボードから1文字の情報を読み取って、それをAレジスタにおさめるサブルーチンは次のようになります。

```

                                *KEYBOARD READ SUBROUTINE
                                *READ ONE BYTE FROM KEYBOARD
                                *CODE IN A REGISTER
4074 3E02      KEYR      ORG      /4074
4076 D38F      OUT      /8F
4078 DB80      KEY1     IN       /80
407A 5604      ANI      4
407C CA7840    JZ       KEY1
407F 3E08      MVI      A,8
4081 D381      OUT      /81
4083 3E01      MVI      A,1
4085 D38F      OUT      /8F
4087 DB80      IN       /80
4089 C9        RET
                                END
                                *STATUS ADRS
                                *READ STATUS
                                *READ COMPLETE
                                *CLEAR STATUS
                                *DATA ADRS
                                *READ DATA

```

これで、データライタの改造とそのインターフェース製作は終わりました。これからはソフトウェアの問題になりますが、この装置を使って会話型モニタを製作した事例は「マイクロコンピュータプログラムの作り方」(産報出版)の第3章に詳述しましたので、ここではそれを繰り返すことはしません。興味のある読者は上記の書物を参考にして下さい。

第4章

高速入出力装置と割込み処理

4.1 はじめに

前章で、マイクロコンピュータに、4つの機能をそなえた入出力タイプライタを接続する回路について述べました。タイプライタの種類はここで述べたものと違っていても、とにかく動作が機械的に行なわれる入出力装置は、処理速度が遅いという点に不満があります。

たとえば、紙テープにパンチされた4KBのアセンブラをメモリに読み込むのに、数十分もかかるというのでは待つほうがいらしてきます。何かの不都合でシステムの一部をこわしたときに、急きょシステムを再読み込みしなければなりません、そういう急ぐときの役に立ちません。こういうときには、機械的な動作にあまり頼らない高速の入出力装置が必要になってきます。

この章では、最初に割込み処理回路の設計法について述べ、その一つの実例として、時計回路（リアルタイムクロック）の作り方について述べます。

次に、ドット型の高速度プリンタ装置のインターフェース回路の製作例を示し、最後に私の研究室で発明した手動式の光電紙テープ読取り装置の作り方について述べます。

4.2 割込み処理

割込み処理の原則的なことについては、既に第2章で検討しました。8080系の割込み手順を再記してみると、CPUのチップの端子に

- (1) 割込み要求 INT (Interrupt request)
- (2) 割込み認知 INTA (interrupt acknowledge)

の2つの端子があって、装置は割込み要求端子に信号を出し、それがCPUによって受け付けられると、割込み認知の端子に信号が出てくるので、それを用いてRST命令をバスに入れる、ということになっていました。

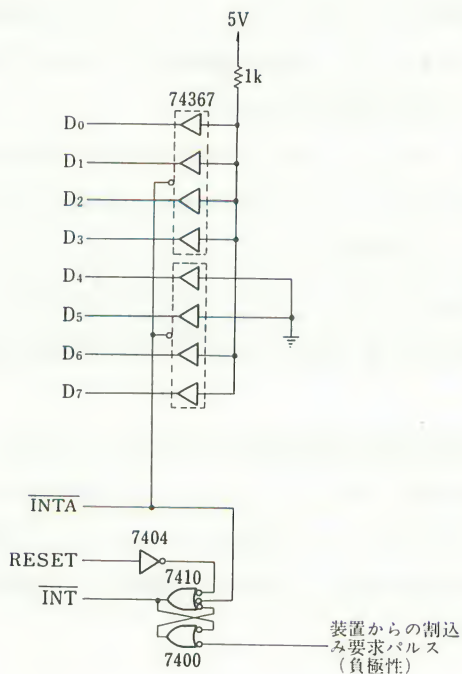


図 4.1 割込み要求回路の基本形

さて、そこで簡単な場合から考えることにして、割込み要求する装置が一つに限られているとします。このときは、割込み要求信号が重複する可能性はまったくないので、図4.1の回路によって目的を達することができます。

この回路で、装置は割込みを要求するときに負極性のパルスを出すものと仮定しています。その要求パルスでフリップフロップをセットし、それによってCPUへの割込み要求線 \overline{INT} をロウに落とします。割込み要求は、CPUによって、ただちに受け付けられるとはかぎらないので、このようにフリップフロップに一時ラッチしておかなければなりません。

CPUがもし割込みを受け付ける状態ならば、この要求は受理され、 \overline{INTA} がロウに落ちます。これで74367のゲートを開いて、データバスにRST命令を乗せます。図のように配線すれば、

RST 1

がCPUに送られるので、コントロールは8番地にジャンプします。8番地に降に、その割込みがきたらどのような処理をするかをプログラムしておいて、処理の最後にRET命令をおくと、コントロールは前に中断した命令のところにいきます。

一般に、マイクロコンピュータで機械装置を制御しようとするとき、割込み発生源は一つとはかぎらず、複数あるのが普通です。こういう場合に、どうしたらよいかあらかじめ考えておかなければなりません。

いま割込み要求源が2つあったとして、それらの要求が時間的にまったく同時には起こらないということがはっきりわかっていたとします。この場合はたとえば図4.2のような回路でよいでしょう。

割込み要求線はワイヤードOR回路を採用し、どちらかの割込み要求が発生したときにロウに落ちるようにします。 \overline{INTA} がCPUから発せられると、割込み要求を出したほうのゲートが開き、バスをドライブしてRST命令を挿入します。図でAの割込み要求は、

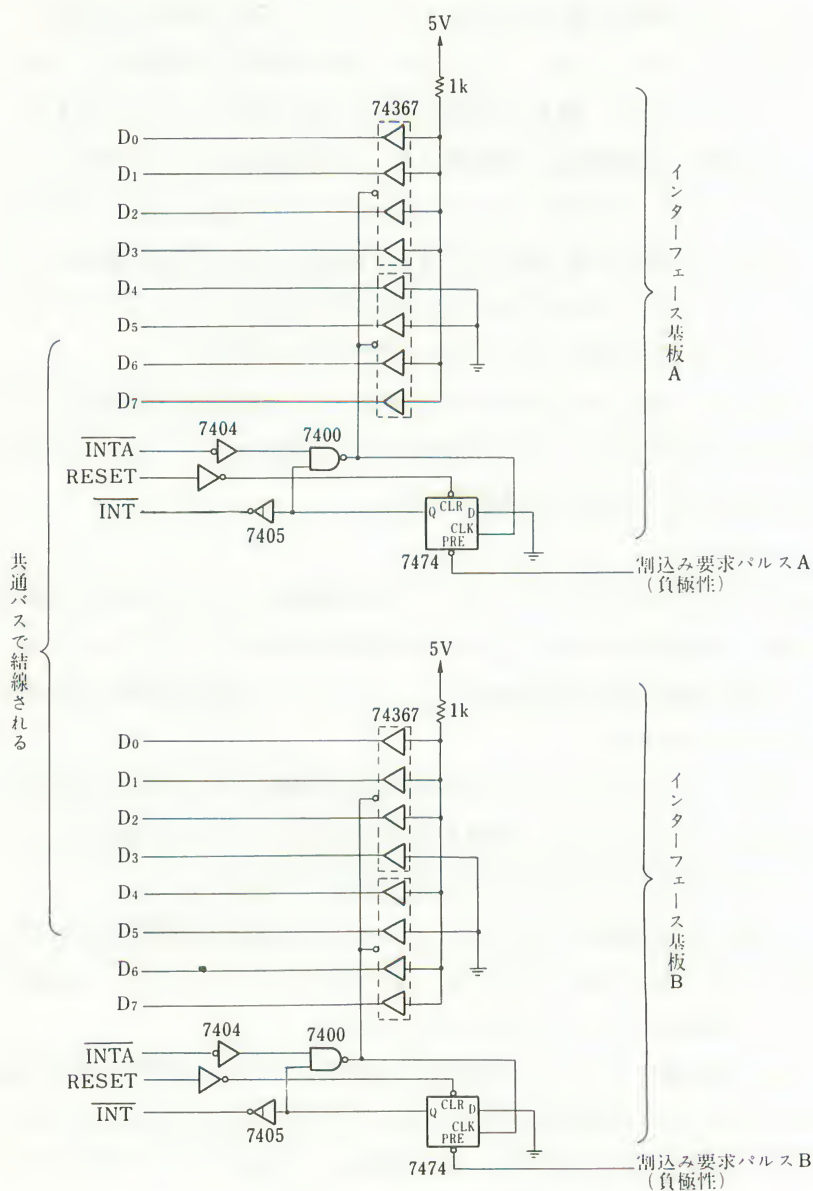


図 4.2 2つの独立な割込み要求の処理 (非重複の場合)

RST 1

を出し、Bの割込み要求は、

RST 2

を出します。したがって、Aの割込み処理プログラムは8番地以降に、Bの割込み処理プログラムは16番地以降に書くことになります。AとBの回路は物理的に離れたところに置くことができますから、マザーボード形式のバスを採用しているときに好都合です。

この回路の不安なところは、何かの条件で、たとえば論理的には絶対なくとも、ノイズの混入などによっても、2つの割込み要求フリップフロップが同時にオンになった場合です。この要求が受け付けられると、両方のRST命令がバスに出てきますから、ドライバの出力回路が損傷を受け、しかも予期しないようなRST命令が入ってしまい、コントロールはとんでもない番地に飛んで行ってしまいます。

この不安を避けるために、図4.3のような回路にしたらどうでしょうか。2つのドライバをやめて、バスドライバは一個所に集めます。もし、Aが割込み要求を出して、Bが割込み要求を出していないときは、INTAに同期して、

RST 1

がバスに乗ります。逆に、Bが要求を出して、Aが要求を出していないときは、

RST 2

がバスに乗ります。これで一応の目的は達成されます。この回路のいやなところは、図4.3で(イ)と(ロ)の線が基板から基板に張らなければならないというところです。とくに、マザーボード形式を取り、ピン数の少ないときに問題になるので注意しておいて下さい。

もし、割込み要求AとBが同時に出たらどうなりますか。これは図4.3を

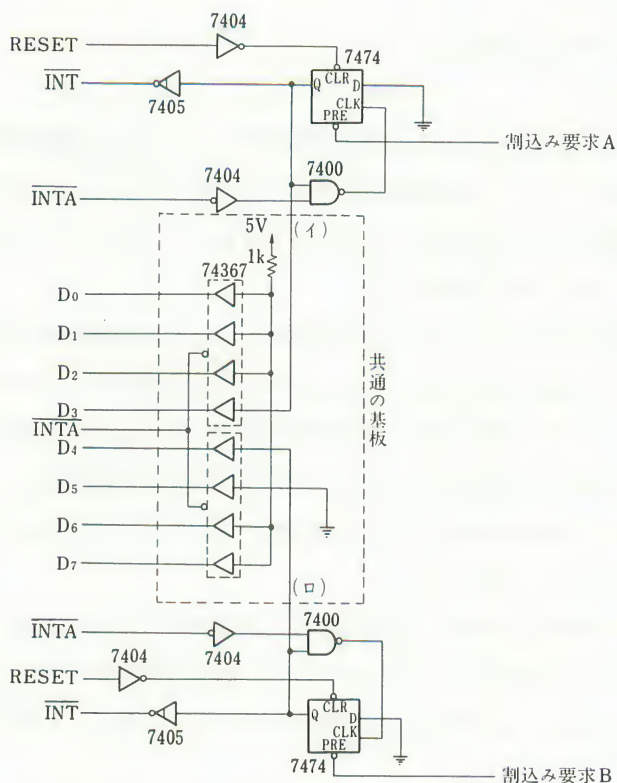


図 4.3 RST 命令を作る回路を一つにまとめた場合

みればわかるように、

RST 3

が CPU に送られます。24 番地以降に A と B の 2 つの割込みが同時に出たときの処理を書きます。

この考え方を押していくと、RST 命令の構造からわかるように、3 つの違った割込み要求源しか取り扱うことができません。図 4.4 に、この場合の概略を示しました。このときの RST 命令と割込み要求の発生状況は表 4.1 のようになります。

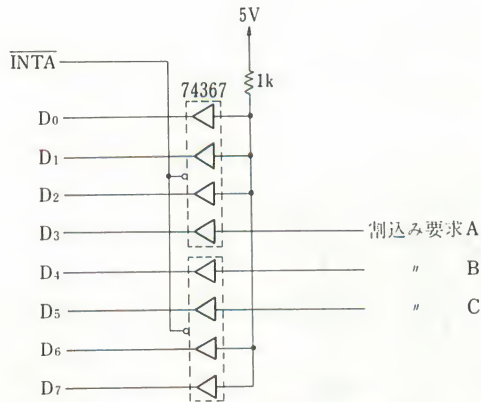


図 4.4 3 個の割込み要求の場合

3 つ以上の割込み要求源があるときには、割込み要求をコントロールすることが必要になります。これにちょうどぴったりの IC がフェアチャイルドの 9318 8 入力優先度つきエンコーダです。この IC の論理回路を図 4.5 に再掲します。オペレーションの真理値表は表 4.2 のようになります。

\overline{EI} の端子は図 4.6 のように GND に落とします。その状態で、もし $\overline{I_7}$ をロウに落とすと、その他の入力の状態に関係なく、

$$\overline{A_0} = \overline{A_1} = \overline{A_2} = 0$$

となります。 $\overline{I_7}$ がハイの状態、 $\overline{I_6}$ をロウに落とすと、 $\overline{I_5} \sim \overline{I_0}$ の端子の状態

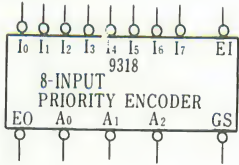
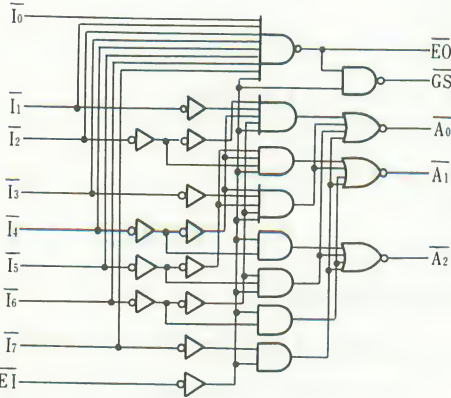
表 4.1 割込み要求の有無と飛び先番地

命 令	割込み要求			飛び先番地
	A	B	C	
RST 1	有	無	無	8 番地
RST 2	無	有	無	16 番地
RST 3	有	有	無	24 番地
RST 4	無	無	有	32 番地
RST 5	有	無	有	40 地番
RST 6	無	有	有	48 番地
RST 7	有	有	有	56 番地

に無関係に

$\overline{A_1} = \overline{A_2} = 0, \overline{A_0} = 1$

となります。以下同様です。



← 図 4.5 8 入力優先度つきエンコーダ (フェアチャイルド)
↓

LEADS

LOADING

$\overline{I_0}$	Priority (Active Low) Input	1	UL
$\overline{I_1} - \overline{I_7}$	Priority (Active Low) Inputs	2	UL
\overline{EI}	Enable (Active Low) Input	2	UL
\overline{EO}	Enable (Active Low) Output	5	UL
\overline{GS}	Group Signal (Active Low) Output	6	UL
$\overline{A_0}, \overline{A_1}, \overline{A_2}$	Address (Active Low) Outputs	10	UL

表 4.2 9318の入出力の真理値表

	\overline{EI}	$\overline{I_0}$	$\overline{I_1}$	$\overline{I_2}$	$\overline{I_3}$	$\overline{I_4}$	$\overline{I_5}$	$\overline{I_6}$	$\overline{I_7}$	\overline{GS}	$\overline{A_0}$	$\overline{A_1}$	$\overline{A_2}$	\overline{EO}
(DISABLED)	H	X	X	X	X	X	X	X	X	H	H	H	H	H
(NO ACTIVE INPUT)	L	H	H	H	H	H	H	H	H	H	H	H	H	L
	L	X	X	X	X	X	X	X	L	L	L	L	L	H
	L	X	X	X	X	X	X	L	H	L	H	L	L	H
	L	X	X	X	X	X	L	H	H	L	L	H	L	H
	L	X	X	X	L	H	H	H	H	L	H	H	L	H
	L	X	X	L	H	H	H	H	H	L	H	L	H	H
	L	X	L	H	H	H	H	H	H	L	L	H	H	H
	L	L	H	H	H	H	H	H	H	L	H	H	H	H

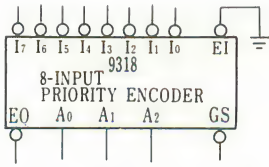


図 4.6 EI 端子は GND に落とす必要がある

だから図 4.7 のように配線すると、8 個の割込み要求がさばけることになります。9318 にはラッチの能力はないので、図 4.7 の割込み要求は、図面に入る前のところでラッチしておかなければなりません。また、そのラッチのリセットも、ハード的に簡単にはいきませんので、各インターフェース

回路のなかで対策を立てなければならないでしょう。

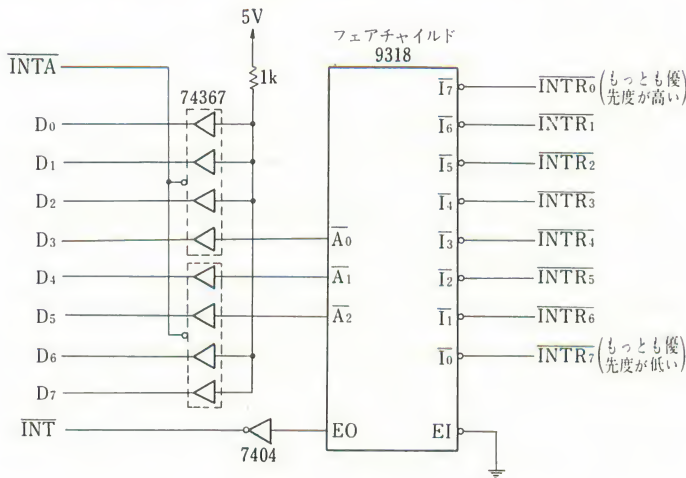


図 4.7 8 入力の優先度つき割込み処理

この方式を採用するときには、図 4.7 の回路は CPU カードのなかに入れるのが良いと思います。割込み要求のための 8 本の信号線 $\overline{\text{INTR}}_0 \sim \overline{\text{INTR}}_7$ が CPU カードの 8 本のピンを占有しますが、それが可能なときには、比較的簡単に 8 個の優先度つきの割込み処理ができます。

ここで優先度というのは、 $\overline{\text{INTR}}_0$ がロウに落ちると、その他を無視して、

RST 0

がバスに乗ることをいいます。 \overline{INTR}_0 がもっとも優先度が高く、 \overline{INTR}_1 がその次で、……というようになって、 \overline{INTR}_7 がもっとも優先度が低いことになります。たとえば、 \overline{INTR}_3 と \overline{INTR}_5 が同時にロウに落ちると、

RST 3

が CPU に送られることになり、コントロールはひとまず24番地に飛び、そこで \overline{INTR}_3 に対する割込み処理を行ない、 \overline{INTR}_3 の要求を解除しますが、その後、CPU が割込み受付け可能の状態になると、 \overline{INTR}_5 の要求はまだ残っているから、次に

RST 5

が入り、40番地以降の割込み処理プログラムが実行されます。

このように、図4.7の回路を採用すると、2つ以上の割込み要求が同時に発生したときは、そのなかでもっとも優先度の高い要求が先に処理されることになります。これが優先度の意味です。

割込み要求源が8を越えたらどうするか、という質問があるかもしれませんが、マイクロコンピュータの場合は、私のこれまでの経験によりますと、たいてい8以下におさまるようです。もし、どうしてもそれ以上になるときは、ミニコンピュータなどで採用されているポーリング方式をとらなければならないと思いますが、そこまでしなくても良いようです。

4.3 時計回路と割込み処理プログラム

これまで2度にわたって、割込み処理について述べてきたので、その仕上げとして、割込み回路と処理プログラムの実例について述べます。

まず最初に、低速バス基板上に作った割込み優先回路を図4.8に示します。優先度のつけ方は前節で述べた方法を採用しました。同時に、ラッチも同じ基板の上に寄せ、装置からは負極性のパルスだけを送るようにしました。割

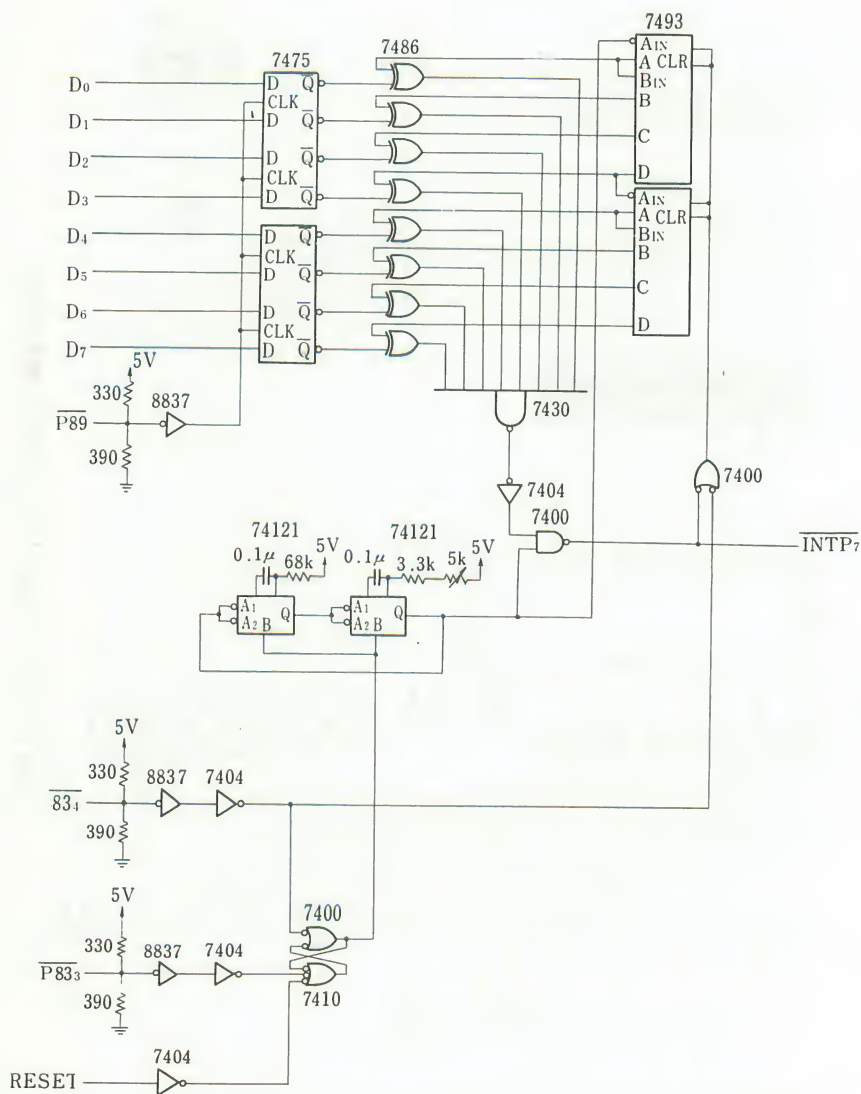


図 4.9 リアルタイムクロック回路

RST 1

が CPU に送られ、8 番地にコントロールがジャンプしますが、そこで

```
MVI    A, 2
OUT    /80
```

とすると、この割込み要求がリセットされます。以下同じです。

以上の準備をしておいて、リアルタイムクロックの製作に入ります。製作した時計回路を図 4.9 に示します。タイムベースとして、2つのワンショットを使い、これで約 1 msec の波形を得るように可変抵抗を調整します。

発振のスタートとストップはフリップフロップによってコントロールされていて、 $\overline{P83_1}$ のパルスによって発振がスタートし、 $\overline{P83_2}$ のパルスによって発振がストップします。

この発振波形がクロックとして、2 個の直列接続された 2 進カウンタに入り、カウンタの内容をカウントアップします。

一方で、OUT / 89 の命令でデータが設定できるラッチがあって、このデータとカウンタの内容が一致したときに割込みパルスが出ます。割込みパルスはカウンタの内容をリセットしてオールゼロにします。クロックのスタートパルスもカウンタの内容をクリアします。

最初に、時計回路の性能を調べてみます。割込み回路のラッチに数 n をセットしておいて、割込みが入るまでレジスタをインクリメントして、割込みが入ったら、その時点でレジスタの内容をデータライタに印刷します。プログラムのリストは次のとおりです。

```

**
* REAL TIME CLOCK TEST PROGRAM
**
SPAD    EQU    /200
PRAD    EQU    /411A
MNTR    EQU    /4000
0000    310002    LXI    S, SPAD
0003    210000    LXI    H, 0
0006    3E01      MVI    A, 1                *DATA N SET
```

```
0008 D389      OUT    /89
000A FB        EI
000B 3E10      MVI    A,/10      *ENABLE INTERRUPT
000D D383      OUT    /83      *START CLOCK
000F 23        UPCT  INX    H      *INCREMENT
0010 C30F00    JMP    UPCT      *REPEAT ENDLESS

**
* INTERRUPT PROCESSING ROUTINE
**

0038 3E08      MVI    A,8        *STOP CLOCK
003A D383      OUT    /83
003C 3E80      MVI    A,/80      *RESET INT REQUEST
003E D380      OUT    /80
0040 CD1A41    CALL   PRAD
0043 C30040    JMP    MNTR      *RETURN TO MNTR
                        END
```

スタックポインタを設定し、HLレジスタをゼロに払い、データ n をラッチにセットし、割込みを受付け可能な状態にして、クロックをスタートさせ、それからHLレジスタを一つずつインクリメントします。

この間に、時計回路から割込みパルスが入るとコントロールは／38番地に飛びますから、そこでクロックをとめ、割込み要求フリップフロップをリセットして、HLレジスタの内容を印刷します。PRADはHLレジスタの内容を16進数4文字に印刷するサブルーチンの名前です。仕事が終わったらモニタに帰り、7番地の数 n を変更して、再び実験します。実験の結果は表4.3のようになりました。

表4.3 HLレジスタの内容を16進数で表わした結果

n	HLレジスタの内容
1	0080 (16)
2	0105
4	0210
8	0426
16	0851
32	10A7
64	2154
128	42AD

これでは内容があまりよくわからないので、まず、HLレジスタの内容を10進数に変換し、それから n で割ってみると表4.4のようになりました。これをグラフにプロットしたのが図4.10です。明らかに、 n の小さいところが、 a/n が小さくなっていることがわかります。

この原因はおそらくクロックの最初の1カウントのパルス幅が小さく出ているのではないかと推察されます。そこで、全部のデータか

表 4.4 レジスタの 1 ビット当たりの
インクリメント回数

n	HL の内容 10 進数 a	$a \div n$
1	128	128
2	261	130.5
4	528	132
8	1062	132.8
16	2129	133.1
32	4263	133.2
64	8532	133.3
128	17069	133.4

ら最初の 1 カウント分を除去して、その後に同じ計算をしてみます。これには、

$$\frac{a - 128}{n - 1}$$

という計算式を使えばよいことがわかります。結果をプロットすると、図 4.11 のようになりました。推定したように、結果はかなり改善されました。

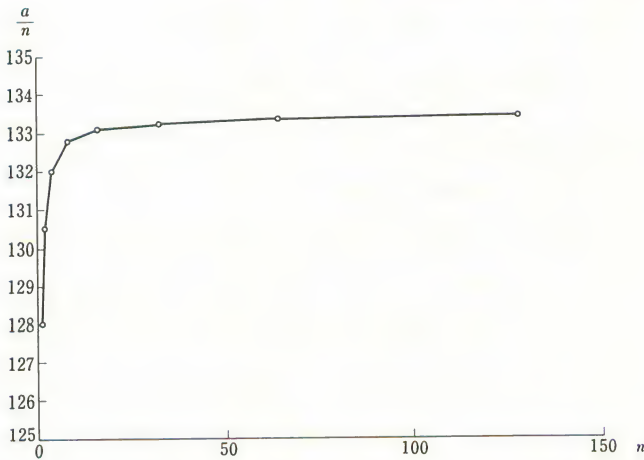


図 4.10 時計回路のチェックデータのプロット

図より、 n のごく小さいところを除いて、1 クロック当たり

INX H

JMP UPCT

を 133.4 回実行していることがわかります。これらの命令のステート数は 5 と 10 ですから計 15 ステートになり、2 MHz のクロックを使用しているので、

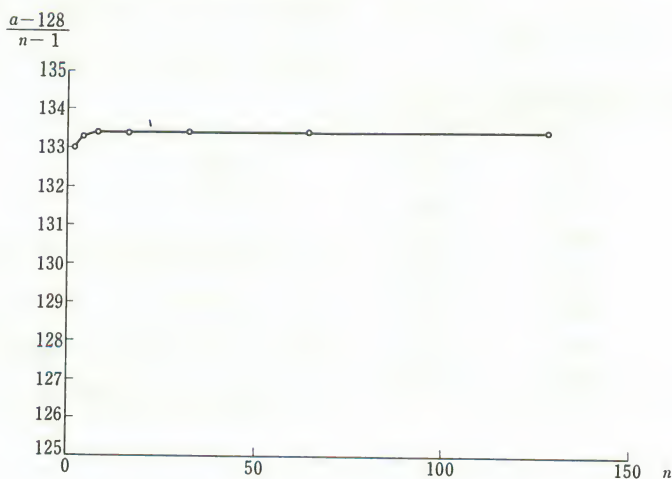


図 4.11 最初の 1 クロックを除いて計算した値

時計回路の波長を推定すると、

$$133.4 \times 15 \times 0.5 = 1000.5 \mu\text{sec}$$

となります。ほぼ 1 ミリ秒になっていることがわかりました。

最後にいくつかの実用サブルーチンを作ります。上の実験の結果より、ラッチにセットするデータ n の数は大きいほうが良いので、0.25 秒の時間を消費してメインルーチンに戻るサブルーチンを作ります。これは次のようになります。

```

**
* REAL TIME CLOCK ROUTINE
* WASTE 0.25 SECOND
**
0200 F5      WS25  ORG    /200
0201 3EC9    PUSH  PSW
0203 323800  MVI     A,/C9      *RET TO /38
0206 3EFA    STA     /38
0208 D389    MVI     A,/FA      *0.25 SEC
020A FB      OUT     /89
020B 3E10    EI
020D D383    MVI     A,/10      *START CLOCK
020F 76      OUT     /83
          HLT

```


0210	3E08		MVI	A,8	*STOP CLOCK
0212	D383		OUT	/83	
0214	3E80		MVI	A,/80	*RESET INT REQUEST
0216	D380		OUT	/80	
0218	F1		POP	PSW	
0219	C9		RET		
**					
* WS25 TEST ROUTINE					
**					
		SPAD	EQU	/200	
021A	310002		LXI	S,SPAD	
021D	0E64		MVI	C,/64	
021F	CD0002	TEST	CALL	WS25	
0222	0D		DCR	C	
0223	C21F02		JNZ	TEST	
0226	C7		RST	0	
			END		

250を16進数に変換すると/FAとなるので、これをラッチにセットして割込み受け可能状態にして、クロックをスタートさせ、HLTします。

ここで時計回路より割込みパルスが入るとコントロールは/38番地に飛びますが、ここにはあらかじめ/C9、すなわちRET命令を入れておきます。こうすると、割込みが入ったとたんにHLT命令の次の命令のところに帰ってきます。そこで、時計をとめて、割込み要求フリップフロップをクリアして、メインルーチンに戻ります。

サブルーチンとしてはこれでよいのですが、0.25秒ではテストすることができないので、このサブルーチンを100回呼び出すテストプログラムを作って、ストップウォッチで時間を計測しました。勿論、約25秒でテストプログラムは終了しました。

この他に、たとえばAレジスタに n という数をもって、あるサブルーチンに飛んでいき、そこで n 秒待ってメインルーチンに帰るというようなプログラムを作ることができるでしょう。これは読者の演習問題とします。

4.4 ドット型高速プリンタのインターフェース回路

ある程度まとまった量の情報を印刷しなければならないときに、前章で述

べたような1文字ずつ印字する形式のタイプライタを使うと時間が長くなります。タイプライタに、たとえば1万文字を連続して印字させることは、機械工学の立場から見て、もともと無理があるように思われます。こういうときには、高速の印刷装置を使うべきです。

一般の計算機システムですと、こういう場合にはラインプリンタを使用するようですが、価格が高いというマイナス面があるので、マイクロコンピュータの入出力装置としては採用できそうにもありません。

最近、ドット型式の高速プリンタが発売されるようになり、印刷の高速性と経済性の両面の折り合いがつく程度になったといわれています。私の研究室でも、さっそくこのドット型の高速プリンタ（写真4.1）を入手して、試験的にも使用してみました。

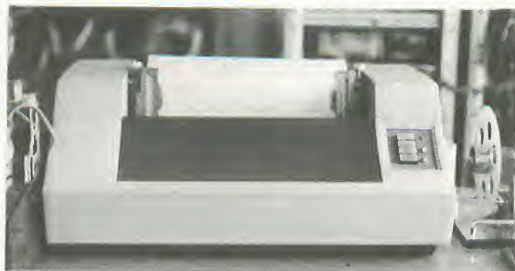


写真 4.1 ドット型高速プリンタ

使ってみてはじめてわかったことですが、第一に印字の際の騒音がかなり激しいこと、第二に文字をドットで構成するために、文字のなかで、たとえば3と8とSのように判別しにくいものがある、という二つの欠点のあることに気づきました。

工業製品の世界では、何かを得れば必ず何かを失わなければなりませんから、価格が安く入手できるというメリットを得たならば、たとえば文字の品質がある程度低下するというデメリットはがまんしなければならないでしょ

う。もし、文字の品質が従来のラインプリンタと同じで、かつ値段が安い製品ができれば、それは従来のラインプリンタは存在価値がないということであって、そうなれば当然、ラインプリンタは産業界から姿を消して行くにちがいありません。

現在の状態がそうっていないということは、得失がバランスがとれているということです。高速プリンタは、コンピュータの出力装置のスペクトルのなかで、タイプライタとラインプリンタの中間層をうめたといえましょう。

最初に、簡単に高速プリンタの印字のメカニズムを紹介します。図4.12は印字に関する駆動機構の概略を図示したものです。

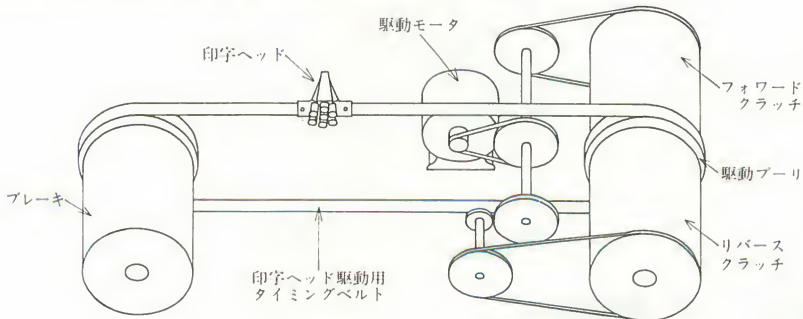


図4.12 印字ヘッドの駆動部の構造

電源を投入すると、駆動モータが回転します。この動力はフォワードクラッチとリバースクラッチを介して駆動プーリに伝達されます。駆動プーリにはベルトが掛けられていて、このベルトに印字ヘッドが固定されています。フォワードクラッチを入れると、印字ヘッドは図の左側から右側に動き、リバースクラッチをオンにすると、印字ヘッドは右側から左側に帰ります。

印字ヘッドには7本のワイヤが縦一列に並べられていて、これがソレノイドによって動くようになっています（図4.13）。図4.14は“H”という文字を印字するときのタイミングを示したのですが、タイミングの1ではすべてのソレノイドが働き、7つのドットが印字されます。タイミングの2, 3,

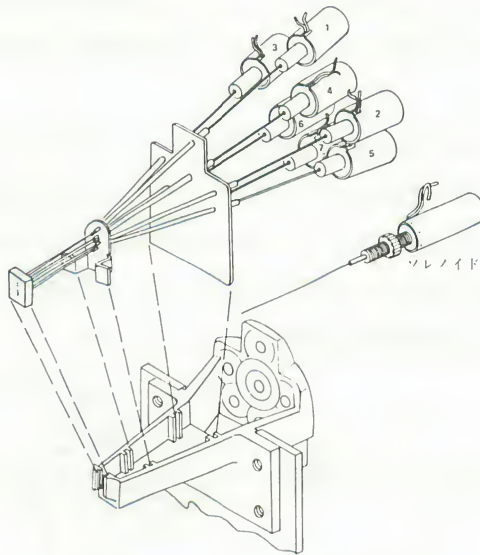


図4.13 印字ヘッドの機構

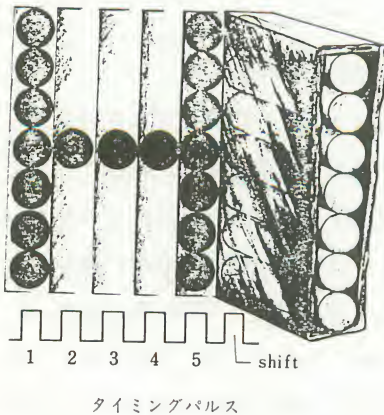


図4.14 文字Hを印字するときのタイミング

4では中央のソレノイドだけが駆動されて横一列のドットが3つ打たれます。タイミングの5はタイミングの1と同じです。これで文字“H”ができました。タイミング5の次に一つのシフトパルスがありますが、これは文字と文字の間を分ける空間になります。機械的な部分はざっとこんな具合になっています。

次に、電気的な構造の概略につ

いて述べます。高速プリンタの制御部は 132 文字分のメモリをもっています。これにあらかじめ 1 行分の文字コードを転送しておきます。文字コードは ASCII コードを採用しています (表 4.5)。

高速プリンタが 1 行の文字の印字を開始するのは

- (a) 132 文字 (1 行いっぱいになる) だけのコードが転送された場合
- (b) キャリッジリターンコードが送られた場合

のいずれかの条件が満足された場合となります。

動作の完了はアクノリッジ (acknowledge) 信号によって CPU に返ってきます。

インターフェース回路の条件について調べてみたところ、まず、文字コードを並列に送るために 8 本の信号線がありました。これを以下で、データ線 (DATA) と呼びます。今回はカナ文字は使用しなかったため、表 4.5 のコードからわかるように、最上位のビットは常時 0 になっています。これはインターフェース基板で GND に結びました。信号線の本数はツイストペアになっているので、 $8 \times 2 = 16$ 本となります。

次に、データをメモリに書き込むためのクロック信号線が 1 本あります。これをデータストローブ線 (DSTB) と呼びます。以上の信号線は CPU が高速プリンタに与えるものです。

高速プリンタの受けのところは、図 4.15 のようになっているので、CPU 側からオープンコレクタゲートでドライブすればよいことがわかります。

データのセットアップ時間、ホールド時間およびストローブのパルス幅の最小値は図 4.16 のように定められています。いずれも $0.5 \mu\text{sec}$ 、あるいは 500nsec です。このパルスは CPU からの信号で直接作することもできますが、こういうところはあまり無理をしないほうがよいですから、データはインターフェース回路でラッチし、またストローブパルスは CPU からのパルスをワンショット回路で幅を広げて使うことにしました。

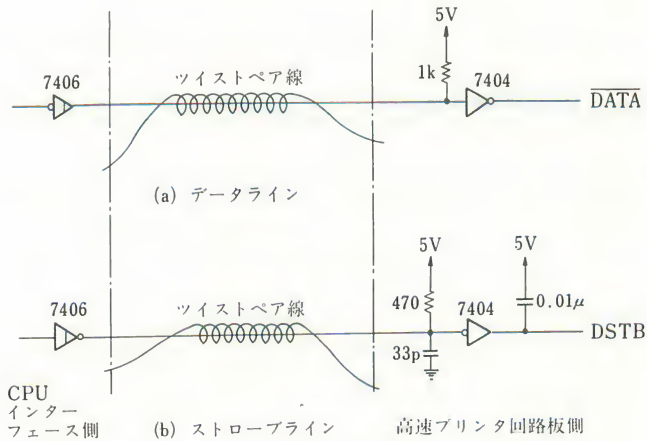


図 4.15 データ転送の回路

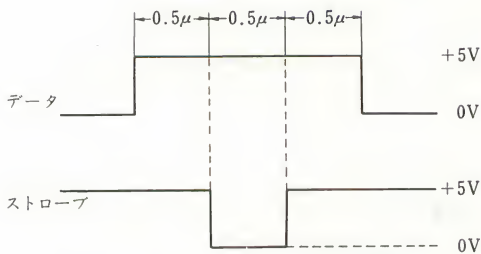


図 4.16 ストロブのタイミング

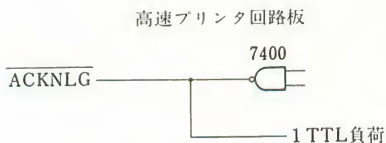


図 4.17 動作完了信号の出力回路

動作の完了信号は図4.17のように、スタンダードTTLゲート7400でドライブされていました。内部で1TTL負荷を既に使用しています。このままではドライブ能力が弱いので、高速プリンタのなかに

増設回路を入れなければなりません、試みにこのままでプリント実験を行いましたところ、それはそれなりに印刷するので、現在のところこのままで使用しています。どうしてここだけこのように

弱い回路を使っているのか理解に苦しむところです。

完了信号のタイミングは文字を送ったとき図4.18のようになります。データストロブパルスの後縁から7μsec後に、4μsec幅のACKNLG信号が来

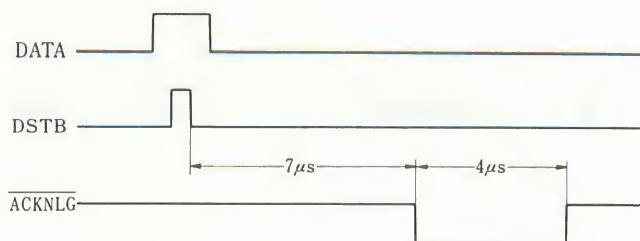


図 4.18 文字を送り出したときのタイミング

ます。この信号の立上りで次の動作に入れば正しい情報の転送ができます。

送り出した文字のコードがキャリッジリターンの場合またはそれが132文字目のときには自動的に文字の印刷が始まります。このときは印刷する文字の数によって印字時間が違うので、何秒後に **ACKNLG** 信号が返ってくるというような確定的なことがいえません。しかし、動作が完了したときに 4 μsec 幅の **ACKNLG** 信号が来ることは同じですから、これを認知する回路を作ればよいことがわかります。

印字動作なしで、紙送りだけするときには、ラインフィードのコードを送ります。このときのタイミングは図4.19のようになります。約75msecの時間が必要なことがわかります。このときも動作が完了すると、4 μsec 幅のパルスが **ACKNLG** として出て来ます。こうしてみると、いずれの場合の、動作が終了すると **ACKNLG** が出て来るという点で統一できることがわかります。

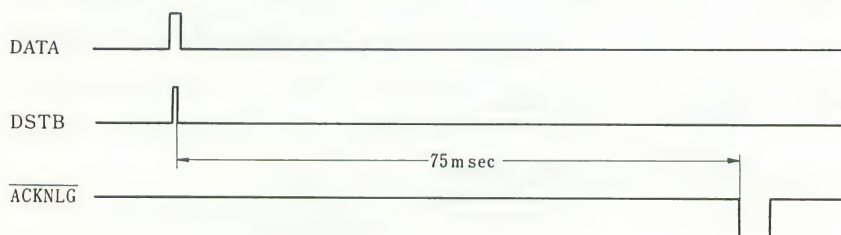


図 4.19 ラインフィード（紙送り）のタイミング

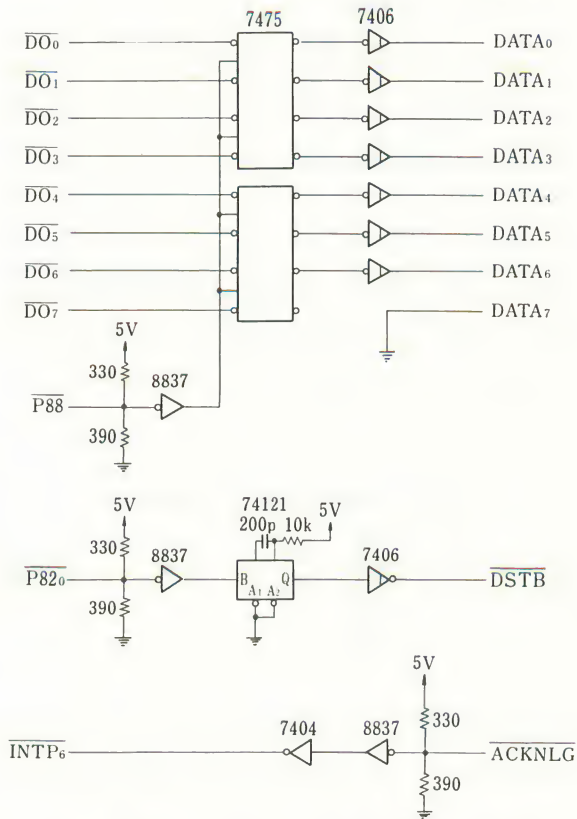


図 4.20 高速プリンタのインターフェース回路

以上の事項を念頭において、図4.20のインターフェース回路を設計しました。文字のコードを

```
OUT  /88
```

でインターフェースのフリップフロップにセットし、

```
MVI  A, 1
```

```
OUT  /82
```

とすることによってコードを高速プリンタのメモリに送り込みます。ACKNLG

信号はそのまま $\overline{\text{INTP}}$ に送りました。

このインターフェース回路を使って1行印刷するサブルーチンを作ります。
まず最初に、割込み処理ルーチンですが、これは/30番地以降に

```
INX SP
```

```
INX SP
```

```
RET
```

という命令を置きます。文字コードを送る段階では印刷は発生しませんから割込みを検知する必要もないと考えられます。1文字を高速プリンタに送るサブルーチンは次のようになります。

```

**
* HIGH SPEED PRINTER SUBROUTINE
**
0200 D388      PRNT      ORG      /200
0202 3E01      OUT       /88          *SEND CODE
0204 D382      MVI      A,1          *STROBE PULSE
0206 C9        OUT       /82
                RET
                END

```

ただし、132文字を高速プリンタに送ることによって印刷を開始させる方式をとる場合はこのサブルーチンは使えないので注意する必要があります。このサブルーチンはコードはAレジスタに入っているものとし、これをまずインターフェース回路のラッチに送り、それからストロブパルスを出します。パルス幅を図面だけから計算すると、単純に

$$0.7 \times 200 \times 10^{-12} \times 10 \times 10^3 = 1.4 \times 10^{-6}$$

となって、約 $1.4\mu\text{sec}$ のパルスが出ることになります。

1行分のコードを高速プリンタに送った後に、キャリッジリターンコードを送って、実際に印刷を開始させます。このサブルーチンは次のようになります。


```

**
* CARRIGE RETURN SUBROUTINE
**
PRNT  ORG    /300
CRLF  EQU    /200
      MVI    A,/40      *CLEAR INT FF
      OUT    /80
      EI
      MVI    A,/D      *SEND CARRIGE RETURN CODE
      CALL   PRNT
      HLT
      END
0300  3E40
0302  D380
0304  FB
0305  3E0D
0307  CD0002
030A  76

```

文字を送ったときの ACKNLG 信号によって割込みフリップフロップがセットされているから、これをリセットし、割込み受け可能な状態にして、キャリッジリターンのコード / 0 Dを送り、HLTします。

ここで、ACKNLG が帰ると / 30番地以降の

```
INX SP
```

```
INX SP
```

```
RET
```

が作用して、このサブルーチンと呼んだルーチンに帰ります。

こころみに、いろいろな文字を 1 行に印刷するプログラムを作ると次のようになります。

```

**
* PRINT ONE LINE ON HIGH SPEED PRINTER
**
PRNT  EQU    /200
CRLF  EQU    /300
MNTR  EQU    /4000
      ORG    /400
0400  310002  TEST  LXI    S,/200
0403  213333          LXI    H,/3333      *INITIALLIZATION
0406  223000          SHLD   /30
0409  3EC9          MVI    A,/C9
040B  323200          STA    /32
040E  0630          MVI    B,/30
0410  48            MOV    C,B
0411  79            MOV    A,C
0412  CD0002  LOOP  CALL   PRNT      *SEND CODE
0415  0C            INR    C
0416  05            DCR    B
0417  C21104          JNZ   LOOP
041A  CD0003          CALL  CRLF      *PRNT ONE LINE
041D  C30040          JMP    MNTR
      END

```


ムにおいて、パンチカードのかわりに使用されてきたものです。メモリの内容をダンプしたり、オフラインでプログラムをパンチしたり、いろいろの用途に利用できます。

紙テープを情報の記憶媒体としたときの欠点は、パンチカードのように、情報の一部を入れ替えるのが簡単でないというところだけです。このところだけがまんですれば、紙テープはコストが安いので、ふんだんに使うことができ、まことに便利です。

紙テープを情報の媒体とするコンピュータシステムを紙テープベースのシステムといいます。これに対して、パンチカードを中心とするものをカードベースの計算機システムといいます。この他に、最近カセット磁気テープをベースにするシステム、フロッピーディスクをベースにするシステムなどもできました。

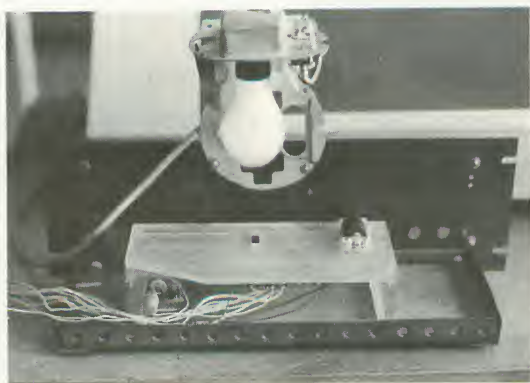


写真 4.2 手動式紙テープリーダの製作

私の研究室で作った手動式光電紙テープリーダの全景を写真 4.2 に示します。写真からもわかるように、装置は簡単なもので、アクリルの板に紙テープが通る溝をつけただけのものです(写真 4.3)。紙テープを押えるために、同じアクリルで作った蓋をします。アクリルの加工は大学の付属工場で行な

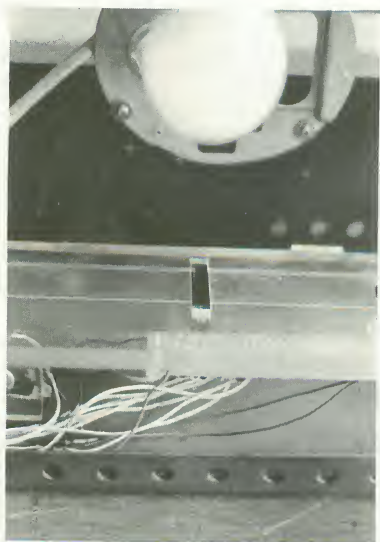


写真 4.3 中央の溝の中を紙テープ
が走る

いました。紙テープが動いているときに進行方向に直交する方向に揺れないようにするために、溝は長くしておきます。

紙テープの穴を検知する素子として、フォトダイオード（またはフォトトランジスタ）が必要ですが、紙テープの穴のピッチが小さいので、物理的に径の小さいフォトダイオードでなければ使用できません。

いろいろ検討してみましたが、細身のフォトダイオードは値段も高いようなので、シャープから発売されている SRC-291 という 9 個のフォトダイオ

ードがカソードコモン状態でモールドされたものを購入しました。図 4.21 はその接続図です。値段はかなり高いと思いました。もっと安価に市場に供給できるようにメーカーは努力すべきでしょう。

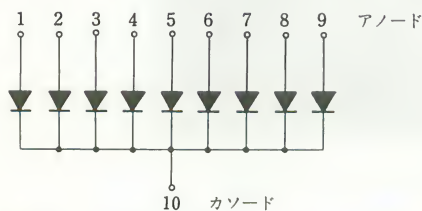


図 4.21 SRC-291の接続図（シャープ）

光源として、コンパクトで横に細長いランプを探しましたが、こういったタイプの光源はどこに行ってもありません。名古屋の市内を足を棒にしてかけずり回りましたが、どの店にもなく、がっかりしてしまいました。

どうしてもだめなので、最後に電気店に行って、100ワットの白熱ランプを買って来て取り付けました（写真 4.2 をもう一度見て下さい）。外観はまこ

とおもしろくないのですが、機能的にはこれで十分です。私のところでは、これを販売して利益を得ようというわけではありませんから、これ以上光源について苦勞を重ねるのは止めました。早く、光源付きのフォトダイオードを発売するように、メーカーにお願いしたいものです。

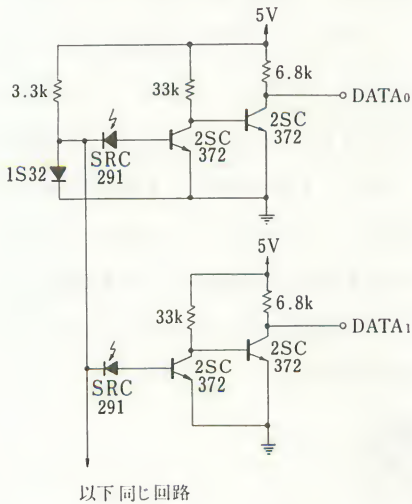


図 4.22 フォトダイオードの増幅回路

まず最初に、フォトダイオードの電流を増幅するアンプが必要になります。これに、図4.22の回路を設計しました。手もとにあった安いトランジスタを使いましたので、値段は安くできましたが、応答特性はあまり良くないようです。光源の高さを適当に調節して、紙テープの穴が来たときに+5ボルト、そうでないときにゼロボルトの出力を得るようにしました。

さて、そこでインターフェース回路の設計に取りかかるわけですが、

まず、紙テープの穴の形状について調べました。私のところで使っている紙テープは8単位の紙テープで、図4.23のように、1ビットの情報を伝達する穴8個と、それから情報があることを伝えるスプロケット1個、合計9個の穴があいています。スプロケット穴はもともと紙テープを繰り出すためのものですが、情報の穴よりも径が小さいので、情報が安定して存在することを検知する信号として使うことができます。

単純に考えれば、スプロケットの信号が1になったときに、アンプの出力をラッチすれば、それで紙テープの上に穴があいているかどうかをフリップフロップの1と0に変換できるということになります。これには図4.24のよ

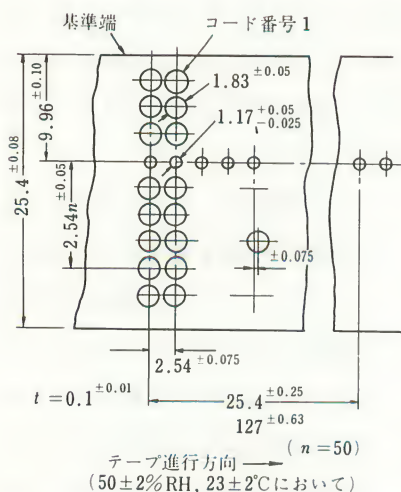


図 4.23 ISOが定めた8単位紙テープの寸法

うな回路でよいでしょう。

CPUの立場からいうと、ラッチの情報をいつ読み込むかが問題になります。これに、シェークハンド方式を採用すると、図4.25の回路になります。

スプロケット信号の後縁でステータスフリップフロップをセットして、これをCPUへのデータラッチの合図にします。CPUはデータを読み取った後に、フリップフロップをリセットしておきます。勿論、データを読み込むパルスを使ってリセットしてもよいのは当然です。

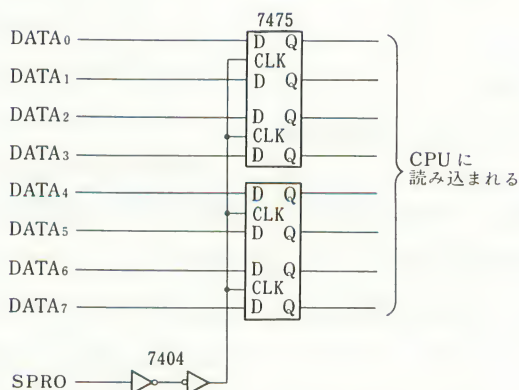


図 4.24 アンプの出力をスプロケットによってラッチする回路

次に、新しい情報がラッチされると、ステータスフリップフロップは1になるので、そのときデータを読み取れば、次の1列の穴の情報がメモリに入

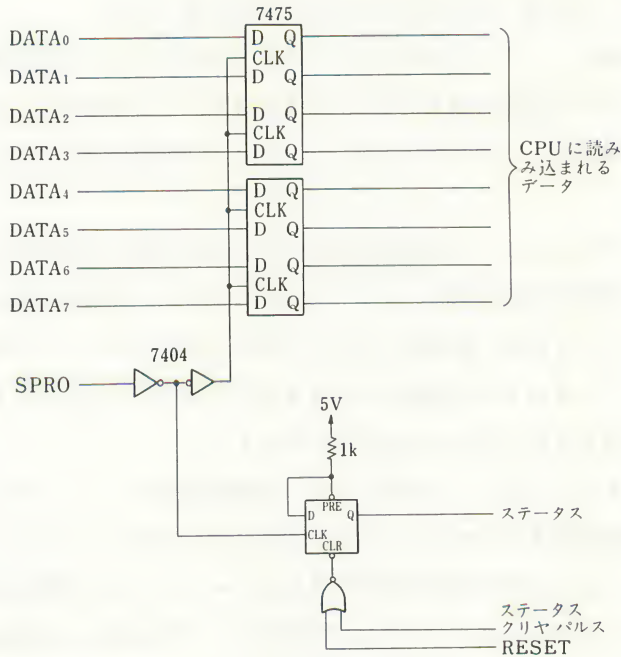


図 4.25 シェークハンドによるデータの転送

ります。割込みを使用する場合は、スプロケット信号の後縁で割込みパルスを作ればよいでしょう。いずれの方法を採用するにしても、簡単なインターフェース回路でデータがメモリに格納されることになります。

この光電紙テープ読取り装置では、紙テープの情報を1文字読んで、それ进行处理し、また1文字読んで処理をするという1文字読取りの機能はありません。それは、この装置には駆動機能がまったくついていないので、人間が手で紙テープを引張らなければならないからです。

人間が手で紙テープを引張っているときに、1文字分ずつ引張れといっても無理なことです。とうていできることはありません。そんな苦痛を人間に与えるくらいならば、こんな紙テープ読取り装置を作らないで、普通の紙

テープリーダを買ってきて使ったほうがよほどましです。

普通の紙テープリーダは高いといってもそんなにびっくりするほどのものではないので、簡単に購入することができます。ここで考えている紙テープリーダは情報の一つのブロックをいっきにメモリの中に読み込むのが目的です。

前もいったように、工業技術の世界では、何かを得れば何かを失うのが常だから、普通の光電式紙テープリーダを買わないで、手動式の紙テープリーダを作ったとすれば、値段は1桁ぐらい安くなるというメリットは得るが、一方で、データの1文字読取りができるというメリットは失うことになります。これはがまんしなければならぬでしょう。

私の考えでは、第一に、市販されている光電式紙テープリーダが1文字読込みの機能を備えるために、かなりやっ介なメカニズムをつけていることと、第二に、メモリにある程度の余裕があればブロックごとに情報を読み込んで何ら差しつかえない、という2つの理由で、1文字読込みの機能のない紙テープリーダを作っても十分役立つと判断したのです。

さて、そういったことで図4.25の回路を実際に製作して、研究室のミニコンピュータに接続したのが、いまから5年くらい前のことです。当時、まだマイクロコンピュータは私の研究室になくて、ミニコンピュータが情報処理の主力になっていました。

このミニコンピュータに、テレタイプがついていましたが、これからアセンブラのシステムテープを読み込むのに30分近く時間がかかり、その上エラーも多くて、うんざりしていたのがこんな装置を作るきっかけになりました。必要は発明の母といいますが、まったくその通りだと思えます。

こういった経験があるので、私は雑誌などでマイクロコンピュータのキットにテレタイプのA S R 33やカシオのタイピュータなどをつけたのを見ると、がっかりしてしまいます。10万円とか20万円のマイクロコンピュータに100

万円もするようなタイピュータをつけて平気な顔をしているようでは、エンジニアの資格は無いというべきです。

マイクロコンピュータにはそれとバランスのとれた入出力装置をつけなければなりません。いま、もし市場にそういった機械がないとすれば、皆で知恵をしぼって新しい入出力装置を考える、というのが正しい態度です。そういった発想の自由がないから、わが国の技術はいつまでたっても、世界の技術の2番せんじだけを汲み取るということになるのです。

閑話休題。それはさておき、こうして製作した紙テープリーダとインターフェース回路を使って、CPUにデータを読込む実験を行ないました。これで100%データが読み込まれたといえ、それで話は終るのですが実際はそう簡単にはいきません。

98%ぐらいの成功率は得られるのですが、それ以上の確実度が得られません。98%というのは、100文字読み込んで2文字ぐらい誤ってしまうということで、これではコンピュータの入出力装置としては失格です。

半年間ぐらい、いろいろな工夫を試みましたが、どうしてもだめです。コンデンサを入れるとかシュミット回路を使うとか、ありとあらゆる対策を試みましたが、すべて失敗におわって最後には、もう打つべき手がなくなっていました。

では、何故失敗したのか、その原因を探ってみると、どうしても成功しない原因は一つ、ただ一つあって、それは人間が手で紙テープを引っ張るとき、そのテープ速度が大きく変動して予想することができないということです。

人間の手はピストンのように往復運動しかしないので、モータのような回転運動と違って、もともとテープを引っ張る速度にむらがあるものです。そのうえ、ときには何かの原因で、テープの動きが止ってしまうようなこともありますし、逆に、あわてて引っ張ると瞬間的にはかなり高い速度になることがあります。だからテープの走行速度を v とすると、これは

$$0 \leq v \leq v_{max}$$

というような形になっていて、 v_{max} は手で紙テープを引っ張るときの限界速度といえます。

まず、テープ速度が大きくなると、フォトダイオードに与えられる光量の変化率が大きくなるので、アンプの出力が追従できなくなります。これは、ちょうどカメラでシャッター速度を速くしたようなものですから、照明を強くするとか、フィルムの感度を増すとか、何か手を打たなければ正常な写真はとれません。

この性能は使用したフォトダイオードの物性によって決まるものですから、何とも対策の立てようがありません。使用者があまり速く紙テープを引っ張らないようにする、というのが唯一の解決策です。

問題はテープ速度が停止も含めて、極端に低くなる場合です。このとき、フォトダイオードのアンプの出力が発振することがあります。

発振の原因ですが、これには2つあって、第一は、テープ上に穿孔された穴の形状ですが、これは肉眼でみるかぎり、なめらかになっているようですが顕微鏡などを使って拡大してみると、切り口にかなり激しい凹凸があって、これによってフォトダイオードに入る光量に変化すること、第二に、テープを引っ張る際にテープがアクリルで作ったガイドにぶつかって、微妙な振動が発生し、これによってフォトダイオードに入る光量に変化する、この2つの原因があります。これらがノイズとなって、アンプがクリティカルな状態になっているときに、振動が発生するものと考えられます。光源に60サイクルの白熱電球を使っていることもマイナーな原因になっているかもしれません。

手で引っ張るというメカニズムを採用し、図4.22のアンプを使う限りにおいては、この発振現象を阻止することはできません。

要は、いかにしてこの発振現象の間をくぐって正しい情報をCPUに読み

込むかです。その方法があるかないかが、この装置の成功と失敗の鍵をにぎっています。

発振現象はアンプのスレッシュホールド近傍で起こるものですから、スプロケット穴の最初と最後のところで発生するに違いありません。これを概念的に書いたのが図4.26です。データの穴の大きさよりもスプロケット穴が小さくなっているので(図4.23参照),スプロケット信号が1になるとき、データの出力は安定しているに違いありません。

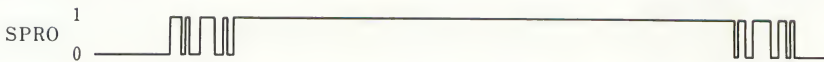


図 4.26 スプロケット信号のチャタリング

だから、図4.25のような回路を使ってデータを読み込む際に、たまたまテープスピードが遅くなると、スプロケットが多重に発生して、何回も続けて読んでしまう、という結果になります。もし、テープ速度が瞬間的にでも限界を超えると、これはスプロケット信号が吸収されて出てこないで、データを読み落すことになります。

いま、テープが停止して、スプロケット信号が発振している状態を考えます。このとき、データは安定しているはずですから、スプロケット信号は図4.27のような波形を出力しています。

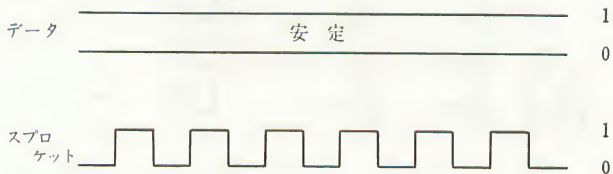


図 4.27 紙テープが停止してスプロケット信号が発振

こういう状況において、唯一度だけ正しい情報をピックアップして、残りの全部のデータを捨てるインターフェースが組めるかという問題ですが、これに対する答えは、残念ながらノーです。このところは、ひとつ皆さんの頭

で、もう一度自分の問題として考えてみて下さい。

紙テープがだいたいどの程度のスピードで動いているかという情報があれば別ですが、そうでないかぎり、テープがとまって発振しているのか、あるいはテープが動いていて同じデータが来ているのか、これを区別する方法はありません。

以上のように考えて、ハードウェアでインターフェース回路を組むことを放棄しました。実際に作ったインターフェース回路は図4.28のようになっています。要するに

```
MVI  A, 8
```

```
OUT  /8 F
```

```
IN   /80
```

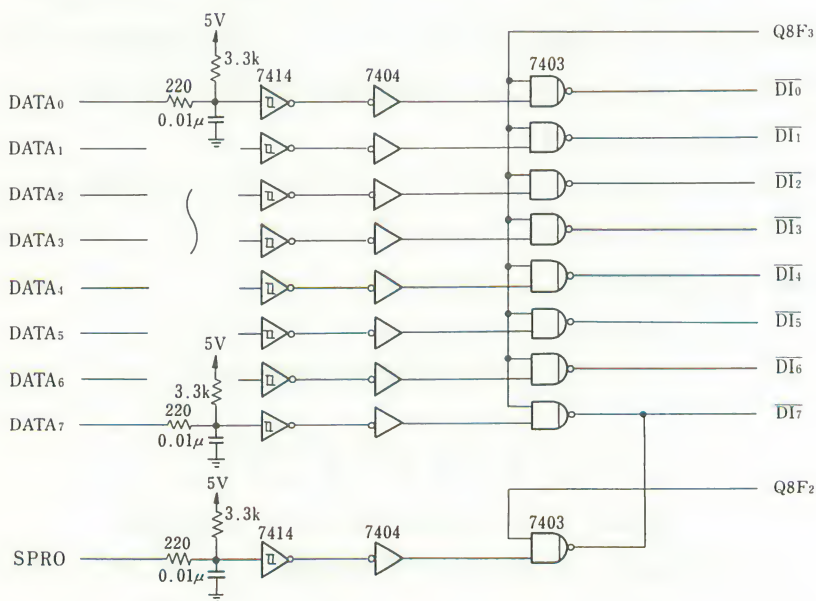


図 4.28 手動式光電紙テープリーダのインターフェース回路

でAレジスタにデータが読み込まれ

```
MVI  A, 4
```

```
OUT  /8F
```

```
IN   /80
```

でAレジスタの最上位 (MSB) にスプロケットの1か0かが入ってきます。
それだけのものです。

こうなれば、あとは知恵だけの問題です。ハードウェアでインターフェースを組むのをやめて、ソフトウェアで処理することに方針を変更したところが成功の大きな原動力になりました。

では、どういうアルゴリズムを作ったか、それを紹介しましょう。最初のヒントは、一つの文字を正しくCPUに送るということだけに考えを集中しないで、文字と文字の関連において正しい情報を採取するという方向に考え方を拡張するということです。

もっと具体的にいうと、図4.23の規格からもわかるように、一つの文字とその次の文字の間にはスプロケットもデータも全てゼロになるところが存在することに気づけばよいのです。この部分に対して、私はブラックという名前をつけました。このブラックを文字と文字の区切りに使うのです。

アルゴリズムはこうなります。

- (1) スプロケットが1になるまで待つ
- (2) スプロケットが1になったら、そのときのデータを読み込む
- (3) ブラックの部分がくるまで待つ
- (4) ブラックが来たことを確認したら、(1)へ戻る

概略フローチャートは図4.29のようになります。

こうして、文字とブラック、文字とブラック、……というように交互に読み込んでいくところがアルゴリズムの核心です。

このアルゴリズムの欠点は、データの8ビットがオールゼロのときに、ス

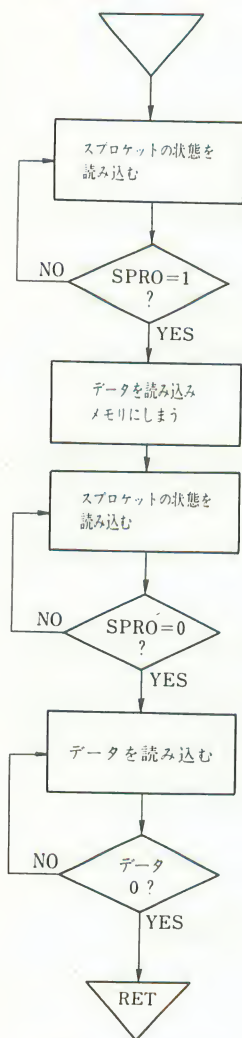


図 4.29 手動式の光電紙テープリーダーから 1 文字読む手順のフローチャート

プロケット信号が発振を起こすと、誤動作する点です。スプロケットが現実には 1 であるのに、発振現象によってゼロになると、そのときにデータがオールゼロのために、アルゴリズムはブラックが来たと認識します。スプロケットが次に発振の 1 になると、新しいデータが来たとして、実は同じデータをメモリに読み込みます。この場合、オールゼロのデータがメモリのなかに入ることは明かです。

このケースが、アルゴリズムが誤認識をする唯一の機会です。これを避けるために次の対策があります。

- (1) パリティを奇パリティにすると、8 ビットのデータのなかに 1 のものが少なくとも一つは存在するので、オールゼロのデータが来ることはない。

テープの先頭と後尾のところに、余白として、オールゼロのフィードがくることがあるが、ここで誤動作してもメモリに読み込まれないので関係ない。

- (2) 発振現象はテープ速度が落ちたときに発生するので、その発振周波数は実験的にだいたい値を測定することができる。

アルゴリズムのなかで、ある時間間隔をおいて、いくつかのデータを採って、それが変化したかどうかをチェックすれば、発振の場合かそうでないかある程度分離することができる。

第 3 章で述べたデータライタは奇パリティ方式を採用

しているので、(1)の条件は満足されます。ただし、バイナリーロードの場合は、あらゆるパターンが来るので、データがオールゼロになることがあります。そこで、(2)の対策は必ず必要になってきます。

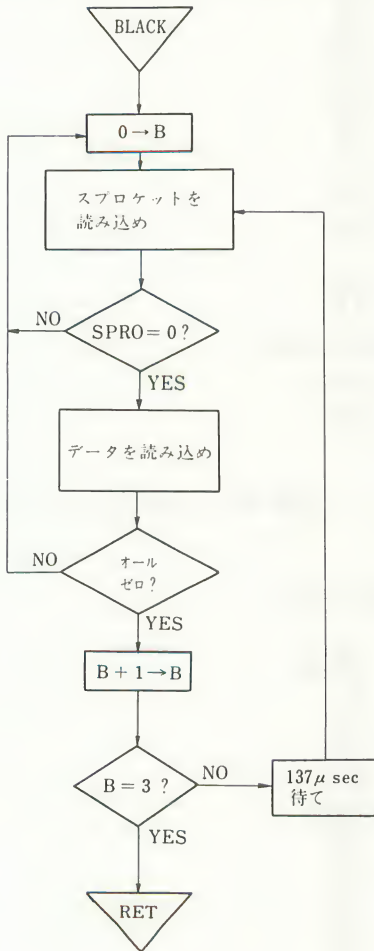


図 4.30 ブラックをみつける手順のフローチャート

これについて、シンクロスコープを使って発振周波数を測定した結果、大略 $137\mu\text{sec}$ おきに3回データを読み込んで、それでオールゼロならば、これはブラックが来たとみてよい、という結論を得ました。

この時間差の設定は非常に微妙ですから注意して下さい。実際の装置を作るときには実験で数値を決めるのがよいと思います。

さて、以上で述べた文字と文字の間隔を確認する手順を詳細フローチャートにまとめたのが図4.30です。実際のプログラムは次のようになります。


```

**
* SUBROUTINE BLACK
* FIND SPACE BETWEEN CHARACTERS
**
                ORG      /2FD8
2FD8 AF          BLCK   XRA      A
2FD9 47          BLCK   MOV      B,A
2FDA 3E04        BLK1   MVI      A,4
2FDC D38F                OUT     /8F
2FDE DB80                IN      /80
2FE0 B7           ORA      A
2FE1 FAD82F       JM       BLCK
2FE4 3E08          MVI      A,8
2FE6 D38F          OUT     /8F
2FE8 DB80          IN      /80
2FEA B7           ORA      A
2FEB C2D82F       JNZ      BLCK
2FEE 04           INR      B
2FEF 78           MOV      A,B
2FF0 FE03         CPI      3
2FF2 C8           RZ
2FF3 CDF92F       CALL     WAIT
2FF6 C3DA2F       JMP      BLK1

**
* WAIT 137 MICRO SECOND
**
2FF9 3E10        WAIT   MVI      A,/10
2FFB 3D          WAT1   DCR      A
2FFC C2FB2F      WAT1   JNZ      WAT1
2FFF C9          RET
                END

```

次に、文字を読み込む手順のフローチャートは図4.31で、実際のプログラムは次のようになります。

```

**
* READ ONE CHARACTER
**
                WAIT     EQU      /2FF9
                CHAR     ORG      /2FB0
2FB0 AF          CHAR   XRA      A
2FB1 47          CHAR   MOV      B,A
2FB2 3E04                MVI      A,4
2FB4 D38F                OUT     /8F
2FB6 DB80                IN      /80
2FB8 B7           ORA      A
2FB9 F2B02F       JP       CHAR
2FBC 3E08          MVI      A,8
2FBE D38F          OUT     /8F
2FC0 DB80          IN      /80
2FC2 4F           MOV      C,A
2FC3 CDF92F      CHR1   CALL     WAIT
2FC6 3E08          MVI      A,8
2FC8 D38F          OUT     /8F
2FCA DB80          IN      /80
2FCC B9           CMP      C
2FCD C2B02F      CHR1   JNZ      CHAR
2FD0 04           INR      B
2FD1 78           MOV      A,B
2FD2 FE05         CPI      5
2FD4 C2C32F      CHR1   JNZ      CHR1
2FD7 C9          RET
                END

```

以上の2つのサブルーチンが完成すればしめたもので、紙テープから1文字読み込むサブルーチンは次のようになります。

```

**
* READ PTR SUBROUTINE
**
BLCK EQU /2FD8
CHAR EQU /2FB0
      ORG /2FA6
2FA6 C5      RPTR PUSH B
2FA7 CDD82F  CALL BLCK
2FAA CDB02F  CALL CHAR
2FAD 79      MOV A,C
2FAE C1      POP B
2FAF C9      RET
          END

```

このプログラムを使って、たとえばヘキサデシマルローダとかバイナリーローダを作ることは読者の演習問題にします。

以上述べた手順は、ぼんやりと読み過ぎせば、ただ単にちょっとしたアイデアを出しただけのものではないか、ということになるかもしれませんが、この考え方は、実はもっと深い思想から発したものであって、単に光電紙テープリーダのインターフェースを作ることに限られたものではありません。人間と機械が出合うときの重要な思想を含んでいます。

機械と機械のインターフェース、たとえばCPUにフロッピーディスクをつけるというような場合は、たとえそれがどんなにむづかしいものであっても、たかが知れています。機械の動作は、たとえそれがどんなに複雑であっても、だいたいきまった範囲のなかで起こるものです。

ところが、機械の相手として人間がからんでくると、問題はとたんにややこしくなります。人間の動作は自由度が大きく、しかも、ときには予想もしなかったようなとんでもないことをしでかすことがあるからです。

このような分野の問題をマンマシンインターフェースの問題ということについては既に述べました。インターフェース回路を作るに当たって、ただ単に論理的な機械装置をマイクロコンピュータにつなげる技術を習得するだけでなく、こういった高度の問題を念頭において、設計を進めなければなら

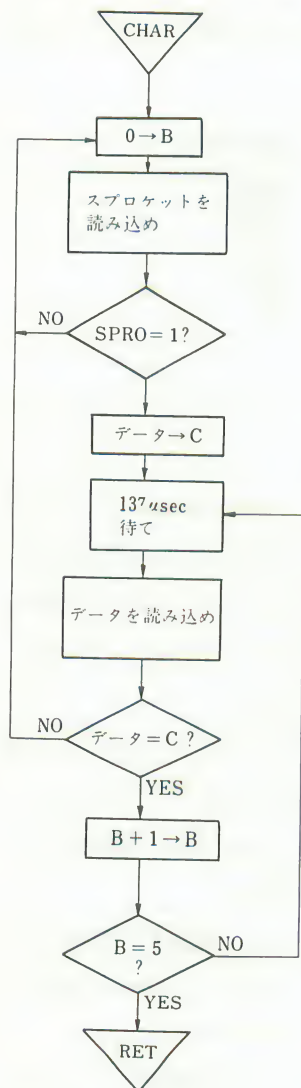


図 4.31 文字を読み込むサブルーチンのフローチャート

ないのだ、ということをご指摘しておきます。

インターフェース技術を皮相的に見れば、デジタル回路の組み合わせの技術とみえるかもしれませんが、実はそうではなくて、もっともって人間臭い技術なのだというのが、はっきりわかると思います。

マンマシンインターフェースの問題を解決するには、人間の心理学あるいは生理学などを含む総合的な学問が必要であって、ただ単に、デジタル回路を作る職人的技能だけでは、とうていそれらの問題を解決し得ないのだ、ということを知るべきです。

マイクロプロセッサのタイミングを理解したり、デジタル IC の使い方に慣れたりすることも確かに必要ですが、そういった低いレベルにいつまでも停滞しないで、常に柔軟で清新な若い心をもって、日々に自己を進歩させる、そういった精進こそ必要なのだといっておきます。

第5章

機械制御のインターフェース回路とDMA

5.1 はじめに

この章では、マイクロコンピュータのアプリケーションの一例として、マイクロコンピュータで電気自動車を制御した事例について述べます。

CPUに8085を採用し、直流電動機とロータリーエンコーダのインターフェース回路の作り方を説明します。最後に、入出力装置の完備した8080Aシステムから、DMAモードで8085のメモリにアクセスする方法について述べます。

5.2 電気自動車の制御

最近、私の住んでいる愛知県で、桃花台という団地と名古屋鉄道の小牧駅を結ぶ交通として、電気自動車を採用するというニュースが新聞に出ていました。このニュースは、おそらく氷山の一角であって、いろいろなところで電気自動車の無軌条車の試作や試験が実施されているものと思われます。

私の研究室では、数年前から、こういった電気で駆動される無軌条車の制御の問題に取り組んできました。勿論、大学で行なう実験ですから、研究の目標はなるべく小さいところに絞っています。昨年は、床の上に白線を描いて、これに追従して電気自動車を走らせたときに、蛇行のような振動現象が

発生したり、それがひどくなると白線からはずれてとんでもない方向に飛び出してしまふ現象を解明しました。その詳しい内容まで述べられるかどうか分かりませんが、順を追って説明していきます。

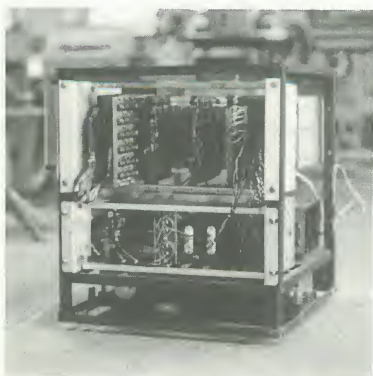


写真 5.1 実験に用いた電気自動車

まず、電気自動車ですが、これは写真 5.1 のようなもので、決して流線型のスマートなものではありません。四角い枠組のなかに、電源とマイクロコンピュータとカメラが積み込まれています。制御特性を調べるには、この程度のもので大丈夫です。

さてそこで、この電気自動車をひっくり返してみると、写真 5.2 のようになっ

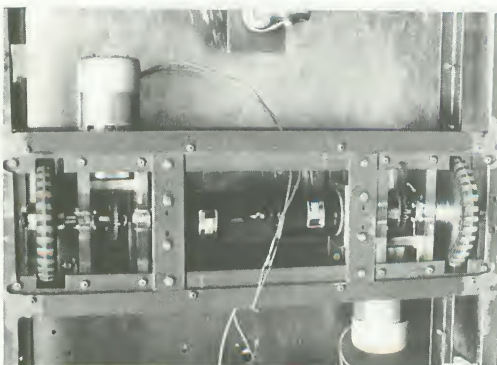


写真 5.2 電気自動車を裏返しにしたときの様子

ています。車輪は4つありますが、普通の乗用車と違って、ダイヤモンド型に配置してあります。ダイヤモンドの各頂点に車輪が一つずつあります。前と後ろの頂点のところの車輪はカスターとなっていて、これは自由に回転できるようになっています。中央の左右

実際の寸法は図 5.1 のようになっています。自動車用語を借りると、シャーシは縦50センチメートルで横50センチメートルの正方形になっています。その前後にカスターがあり、左右の駆動用のモーターがあります。車輪の回転

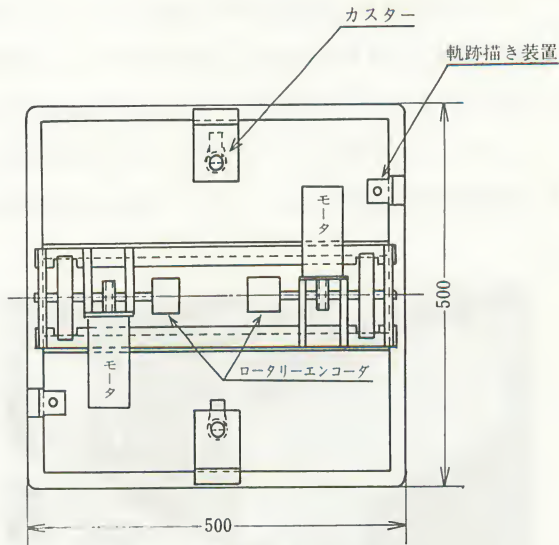


図 5.1 電気自動車の平面図

角を計測するために、光電式のロータリーエンコーダが取り付けられています。

右の車輪と左の車輪はまったく切り離されていて、各々モータによって独立に駆動できるので、このモータをどのように回転させるかによって、電気自動車をいろいろな方向に進めることができます。これは、ブルドーザがキャピラによっていろいろな方向に走るのと同じ原理です。

左右のモータを同じ方向に同じ速度で回転させると、電気自動車は直進しますが、たとえば左のモータを停止して右だけを動かすと、左の車輪を中心にして回転運動をします。その他、左右のモータの電圧を調節することによって、いろいろな走行が可能です。

さて、機械部分はそれくらいにしておいて、マイクロコンピュータですが、CPUに8085を採用し、2.3節で述べたようにCPU基板を製作しました。

8085はTRAPというマスク不可能な割込みがありますが、これをフロント

パネルのスイッチに直結して、電気自動車が暴走したようなときに、このTRAPのスイッチを押すと、それまで電気自動車がどのような状態になっていても、必ず、急停止するようにしました。マスク不可能な割込みは、通常のコンピュータシステムですと電源ダウンなどのときに利用すると良いといわれていますが、機械制御の場合には、このように緊急事態の脱出用に使うことができます。

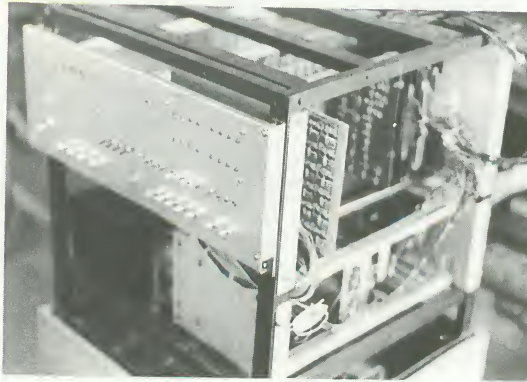


写真5.3 フロントパネルとカードラックの取付け

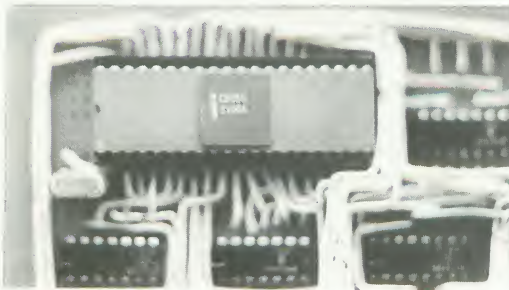


写真5.4 CPUカードのなかの8085のチップ

写真5.3はフロントパネルの様子を写したものです。

写真5.4はCPUカードのなかで、8085のチップまわりのところの写真です。クリスタルには6MHzを使用しました。

5.3 直流モータのインターフェース回路

電気自動車の駆動用直流モータのインターフェース回路をブロック図で書くと図5.2のようになります。8085のバスから出力を2つの8ビットレジスタにラッチし、これをDA変換回路を通してアナログデータに変換し、電力を増幅して直流モータをまわします。

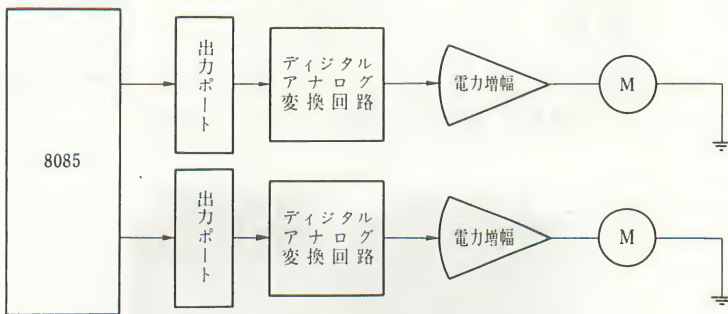


図5.2 インターフェースのブロック図

直流モータとして、沢村電気工業(株)のMM40シリーズのなかで30Wのものを2個使用しました。これはパーマネントマグネットタイプです。供給電圧は24ボルトなので、電力増幅回路は1アンペア強の電流を流すことができればよいことになります。

電力増幅用のサーボアンプとして、図5.3の回路を設計しました。差動増幅回路を2段に使い電力を増幅します。アンプ回路の説明をするのが本書の目的ではないので、この回路の設計プロセスを説明することは省略しますが、こういう点に興味のある読者は書店でオーディオ用のアンプの作り方について書いてある本を参考にしたらよいと思います。オーディオ用もサーボ用も本質的なところはそれほど変わりません。あとがきで述べた(33)の文献は推薦できます。製作したアンプは写真5.5のようになりました。

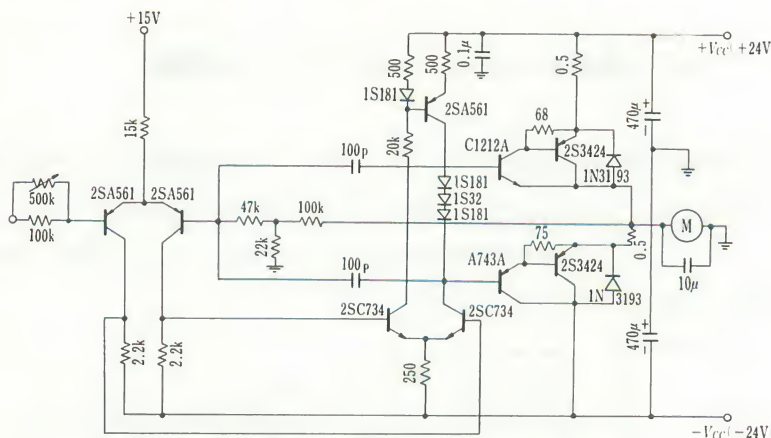


図 5.3 使用した直流サーボアンプの回路

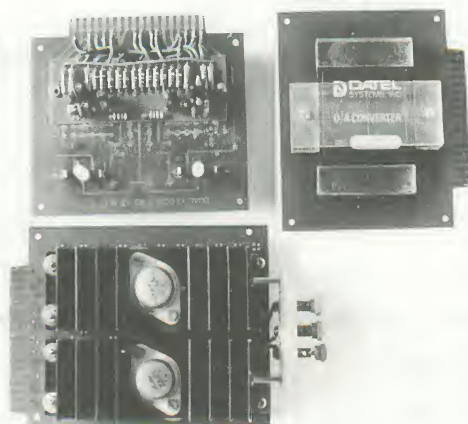


写真 5.5 製作したアンプとDA変換回路

アンプの周波数応答を測定したところ図 5.4 の結果を得ました。ゲインは約 15 デンベル弱ぐらいですが、 10^3 Hz ぐらいから位相遅れが大きくなることがわかります。モータの応答特性は、とてもこんなにはないので、まあまあの結果が得られたものと判断しました。

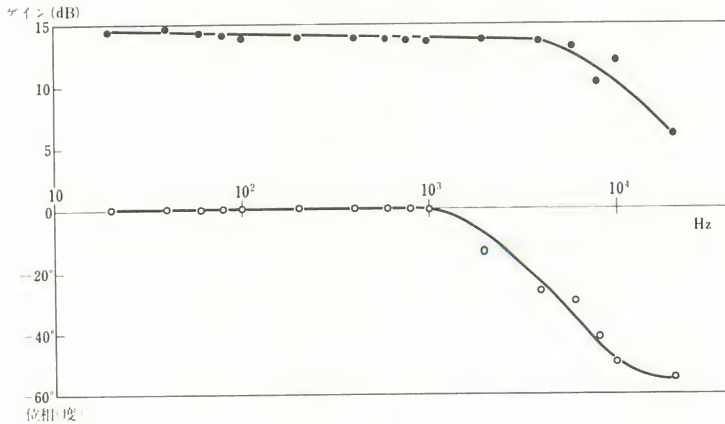


図 5.4 製作したアンプの周波数応答の測定結果

オーディオ用の場合は相手がスピーカーですから、周波数特性が問題になりますが、機械をコントロールする場合は、1秒間に1000回の周期で何かを動かすというのはとうてい無理だから、周波数特性はあまり問題になりません。しかし、そういう時でも、アンプはアンプとして、ある程度の性能を出しているかどうかをチェックするためには、周波数特性はよい指標になります。

これは、とくに述べる必要はないと思いますが、周波数特性というのは入力に正弦波(サイン波)を入れたときに、出力波形を測定して、入力波形の振幅にたいして出力波形の振幅が何倍になったかをゲインといい、入力波形にたいして、出力波形が何度遅れるかを位相遅れといいます。

グラフの横軸を入力波形の周波数にとり、縦軸にたいして出力のゲインと位相遅れをプロットすると周波数応答曲線が得られます。

アンプのなかで波形が歪まなければ、入力に正弦波を入れると、出力にも正弦波が出てきます。そういう回路を線形特性をもった回路といいます。周

波数特性は、実は入力信号の振幅に関係するのが普通であって、一般に、入力信号の振幅の小さい範囲内では線形の特性が得られますが、入力信号の振幅を大きくすると、入力に正弦波を入れても、出力波形が歪む傾向をもちます。これには注意したほうがよいと思います。

アンプにモータを接続したときは、1 kHz などという速い信号は入れることができないので、低周波発振器を使って特性の測定を行ないます。もし、そういった特別の低周波発振器がなければ、コンピュータに接続して、プログラムでゆっくり電圧を上げていけば、近似的な周波数応答を測定することができます。

さて、マイクロコンピュータとアンプを接続するために、ディジタルアナログ変換回路（DA変換器）が必要になります。現在では、かなり安価なDA変換回路が市販されているので、これを自作することはほとんどないと思われるので、その中味の回路にふれるのは省略して、機能的なことだけについて簡単に述べると、まずDA変換回路はマイクロコンピュータのなかで発生したディジタル数値を一つの連続的な電圧に変換する回路と定義できます。

DA変換回路の入力は0か1かの信号線 n 本であり、出力は連続的な電圧を出力する信号線1本です。この他に、当然電源線がありますが、これはとくに述べる必要はないでしょう。

いまマイクロコンピュータのバスが8ビットなので、8ビットのDA変換回路を選ぶとします。このときは、DA変換回路の入力信号線は8本です。数の表現法が2進法か10進法かどうかなのか、最下位ビットと最上位ビットがどのピンに対応するか、というようなことを考えて配線を行ないます。

今回は、DATEL社のDAC-298Bを2個使用しました。この外観は写真5.5に出ています。入力信号は2進法を採用しています。回路のブロック図は図5.5のとおりです。ピンの1から8に2進数を入力するのですが、その際に1

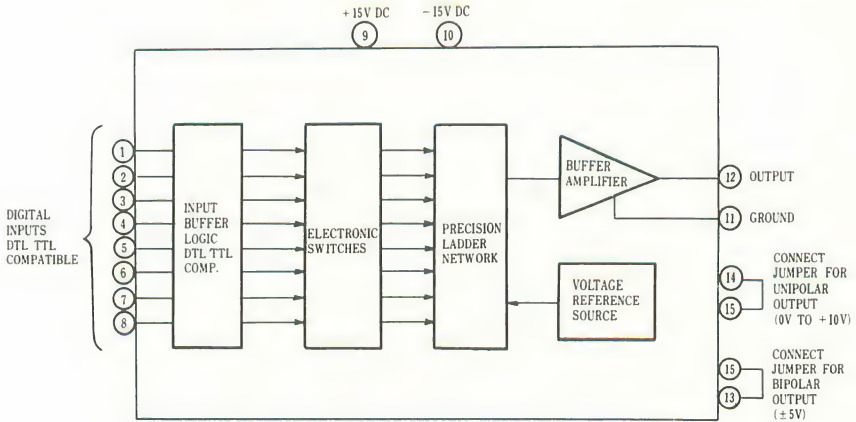


図 5.5 DAC-298Bのブロック図

表5.1 ストレートバイナリー

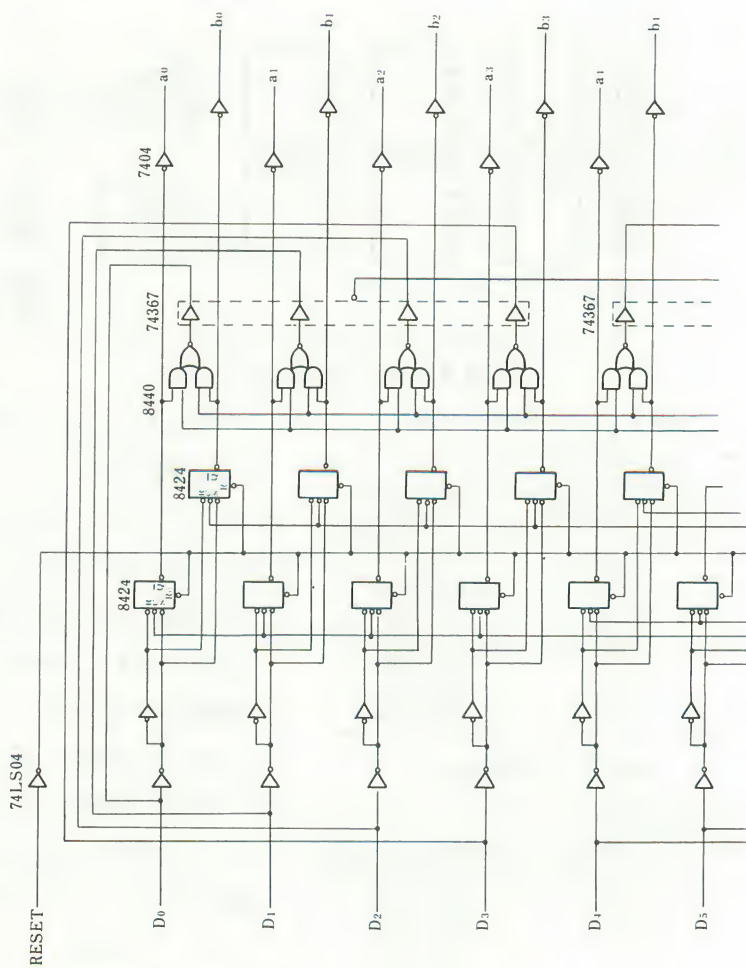
ANALOG OUTPUT RANGE (0 V TO +10V, FS)	STRAIGHT BINARY
+9.96	1111 1111
+8.75	1110 0000
+7.50	1100 0000
+5.00	1000 0000
+2.50	0100 0000
+1.25	0010 0000
0.00	0000 0000

表 5.2 2の補数表現

ANALOG OUTPUT RANGE (± 5 V, VS)	BINARY (ONLY) 2'S COMPLEMENT
+4.94	0111 1111
+4.37	0111 0000
+3.75	0110 0000
+2.50	0100 0000
0.00	0000 0000
-2.50	1100 0000
-3.75	1010 0000
-4.37	1001 0000
-4.96	1000 0001
-5.00	1000 0000

ピンがMSBで8ピンがLSBです。

だから、DACの1ピンに、マイクロコンピュータのバスの D_7 を対応させ、8ピンに D_0 を対応させるように配線します。出力は12ピンから出てきますが、10ボルトの変化の範囲が得られます。もし、ピンの14を15に接続すると入力と出力の関係は表5.1のようになり、ピン13をピン15に接続すると表5.2のような対応関係が得られます。今の場合、モータを正逆に回転する必要があるので、ピン13をピン15に接続して、表5.2の出力が得られるようにしました。



製作したインターフェース回路は図5.6のようになります。まず、2つの8ビットのラッチがあります。これには前に買ったフリップフロップシグネチックス8424の使い残りが沢山あったので、手数が少々余計にかかるのですが、これを使うことにしました。そういう事情がなければ、このラッチは74175にしたほうがよいと思います。ポート番号は／80と／81を割当てました。たとえば

```
MVI A, /40
```

```
OUT /80
```

とすると、一方のラッチに／40がセットされますが、これがDA変換回路でアナログ電圧に変換されて約+2.5ボルトの電圧になり、これが更にアンプで増幅されて、約13ボルトぐらいになります。これが直流モータにかかる電圧です。同じように

```
OUT /81
```

とすると、もう一方のラッチにAレジスタの数値がセットされ、それに対応する直流モータが回転することになります。ラッチにセットされている内容は、おのこの

```
IN /80
```

および

```
IN /81
```

でCPUに読み込めるようにしました。電源の投入の直後に、モータがまわって、電気自動車が動き出すとまずいのでRESET信号でラッチを初期クリアします。

さて、このインターフェース回路を作って、マイクロコンピュータ8085からラッチにデータをセットし、そのとき、モータにかかる電圧を測定しました。その結果は図5.7のようになりました。

入力と出力の関係はほぼ直線の上に乗っていますが、10進数で約110ぐら

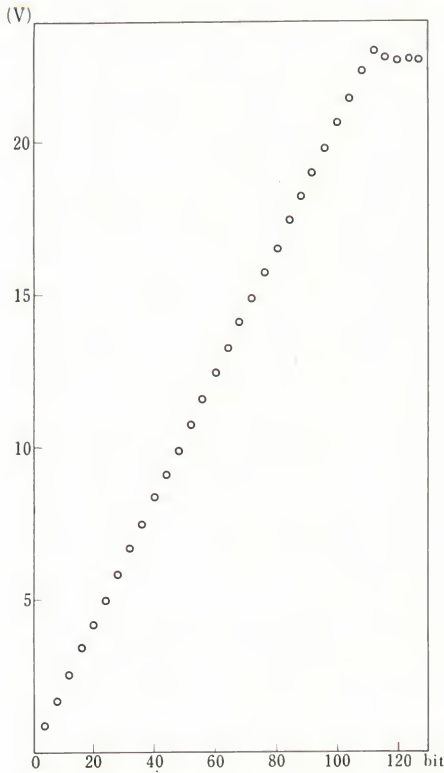


図 5.7 ラッチにセットしたデータとモータにかかる電圧の関係

いの値をセットすると、ほぼ電圧のピークに達し、それ以上ラッチのデータを増しても、出力電圧は増加しないで、図からもわかるように、かえって減少するような傾向を示しました。これはおそらく、アンプの供給電圧を±24ボルトにしてあるので、この電圧に近づくと出力が飽和してくるためだと考えられます。

ラッチにセットできる数値は+1～+127までだから、アンプのゲインをもう少し低くすると全範囲にわたって直線性が得られるかもしれませんが、実際には、アンプのゲインの調整がかなり微妙なので、この辺で手を打つことにしました。

同様にして、今度はラッチに負の数値を与えてみました。DA変換回路が2の補数表現を採用しているので、-1にするには

1111 1111 (／FF)

-2は

1111 1110 (／FE)

……、-128が

1000 0000 (／80)

というようにデータをセットします。正の場合が+1～+127だったのにな

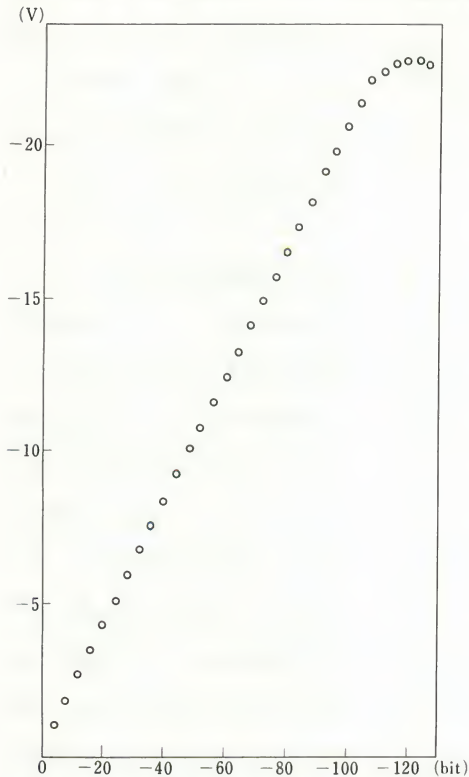


図5.8 ラッチにセットした負の数とモータにかかる電圧の測定値

いして、負の場合が $-1 \sim -128$ となって、一つだけ余分に数値がセットできます。これは負数を2の補数で表現したことの結果です。測定結果は図5.8のようになりました。データはきれいに直線の上に乗っていますが、やはり -110 ぐらいから飽和特性があらわれ直線性が急速に失われます。飽和した後の形状が正と負の場合で違っていますが、その原因はよくわかりませんでした。

最後に、電気自動車をプログラム制御して、実際に地面の上を走らせることにしました。プログラムが誤って自動車を暴走させると危険なので、写真5.6

のように箱を作って、電気自動車をこの上に乗せて車輪を浮かせた状態でプログラムのデバッグをすることにしました。この箱をわれわれは車庫と呼びます。

更に、もう一つの事故対策として、直流アンプの電源を別スイッチにして独立にオンオフできるようにしました。

一番最初のプログラムデバッグは電気自動車を車庫のなかに入れ、かつアンプの電源を切って実施します。このときに、DA変換回路の出力をシンクロスコープで見ると、プログラムが正しく動いているかどうか見当がつきま

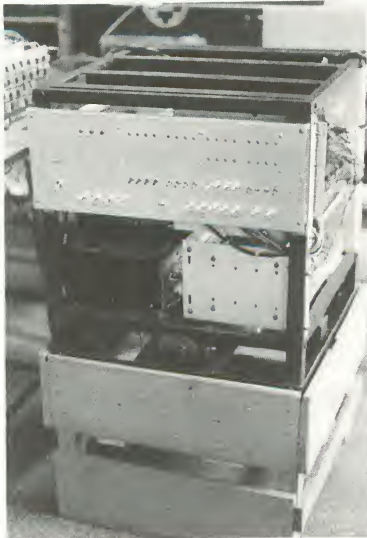


写真 5.6 箱の上に乗せてテストを行なう

て計画を進めます。

数年前に、ある学生が電気自動車の前にすわって、コントロールパネルからメモリのプログラムを修正していたときに、突然電気自動車が走り出して、学生にぶつかって前歯を何本か折ってしまったことがあります。

こういう事故を防止するために、実験の実施に当たって、管理を厳しくするようにいい渡すのですが、学生は社会経験が不足しているためか、注意が散漫で、時々事故が発生するのはまことに残念なことです。このときも、事故の原因はスイッチの押し間違いだろうと思いますが、本人もはっきり記憶していないというので、今となっては事故の原因を究明することはできません。

人間のミスには、プログラムの間違いと操作上のミスの2種類がありますが、プログラムの間違いは最初は当然あるべきものということがわかってるので、これにはたいいていの人がそれなりの対策を立ててのぞむからよいの

す。そこでアンプの電源を入れてもう一度プログラムを実行すると、今度はモータが回転するのでストップウォッチなどを使って計測すると、プログラム制御がうまくいっているかどうかが略々判定できます。それでよければ、最後に電気自動車を車庫から床の上におろして、そこでマイクロコンピュータをスタートさせます。このように、実験を3段階に分けて進めることにしました。

マイクロコンピュータで機械を制御する場合、もし誤った動作が発生すると危険なので、このように慎重に段階を追っ

ですが、プログラムが正しくなって、それを実行する段階で起こる操作上のミスは、まったくランダムに発生するミスなので、どうして防いだらよいかわからないことが多くあります。こういう点を解決するのがマンマシンインターフェースの役目の一つだと思いますが、現実にはあまり顕著な成果はあがっていないようです。

多くの学生と一緒に実験を進めてきて感ずることは、奇妙なことなのですが、事故を起こし易い人とそうでない人がいるということです。とにかく、事故を起こしてけがをしたり装置をこわしたりする人は繰り返しそういう事故を起こすのに、一方で全然そういう事故を起こさない人がいるのです。

これはおそらく、心理学の分野の問題になるのであって、われわれ工学の分野に従事するものが口を出すべき問題ではないのかもしれませんが、とにかく人、その人個人がもっている本来的な力にいくらか差があると考えなければならぬようなふしがあります。

その差は、個人の生い立ちとか教育プロセスとかいろいろな原因によって形成されるのでしょうが、できることならば、人間の頭脳は緻密な判断ができるように、小さい時から訓練しておくべきでしょう。

それはさておき、電気自動車の走行をプログラム制御する方法について述べると、これは、結局、／80と／81のラッチにどのような数値パターンを時系列に入れていくか、ということになります。このとき注意しなければならないことは、周波数応答のところで述べたように、あまり速く数値を変化させると直流モータが追従できないということです。

では、どのような速度で数値を変化させることができるかということです。これにたいして理論的に追究するのはむづかしいので、実験的に調べることにしました。最初はゆっくりとデータを変化させます。それで何ともしなければ、次にもう少し速くデータを変化させて自動車を動かします。

こういうプロセスを繰り返して、安全を確かめながら、ステップを進めま

したが、実験の過程で、直流アンプが発振して出力のパワートランジスタが破損する事故が何回かありました。

そういうにがい経験を通して、現在では0.1秒ごとに1ビットを増減する、

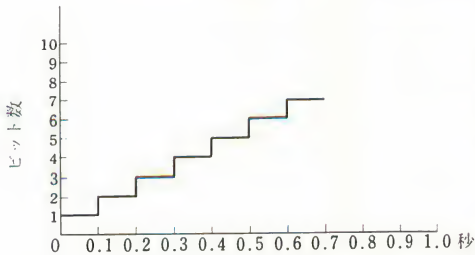


図 5.9 0.1秒に1ビットの増加の場合

というのを一応の規格にしています。だから、たとえば停止している電気自動車を起動するときに、図 5.9 のように、階段状に電圧を上げていくことになります。動いているモータを停止するときもまったく同じです。

これより遅いピッチ、たとえば1秒に1ビット増減するのは安全側ですから問題はありせん。こういうルールを確立した後は、パワートランジスタが破損するというような事故はほとんどなくなりました。

以上の準備をして、モータを回転するプログラムを作りますが、入力波形は図5.10の三角波形とし、1ビットを増減する時間間隔は安全をみて1秒としました。1秒の時間待ちをするプログラムが必要ですが、これには実は第4章で述べたのとはほとんど同じ時計回路を8085用に製作して使用しているのですがそれをここで再述するのはつまらないので省略して、なるべくいろいろな手法を紹介するという意味で、ここではプログラムで時間をつぶすルーチンを作ります。

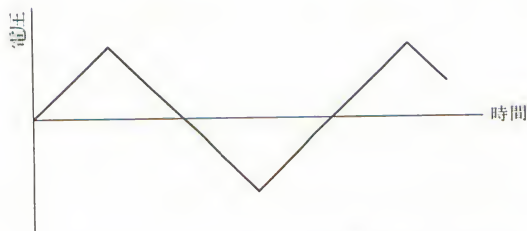


図 5.10 入力波形として三角波形を入れる

170 第5章 機械制御のインターフェース回路とDMA

まず、最初に次のプログラムを作ります。

```

**
* WAIT 3.43 MSEC
**

0200 F5      WAIT  ORG    /200
0201 AF      PUSH  PSW
0202 321102   XRA    A
0205 3A1102   STA    WS
0208 3C      WAIT  LDA    WS
0209 321102   INR    A
020C C20502   STA    WS
020F F1      JNZ    WAIT
0210 C9      POP    PSW
0211         RET
                DS    1
                END

```

このプログラムはWS番地を最初ゼロにして、それから1ずつインクリメントしていった、再び0になったらもとに帰るので、その時間を計算すると、8085ユーザーズ マニュアルから

PUSH	PSW	12ステート
XRA	A	4ステート
STA	WS	13ステート
LDA	WS	13ステート
INR	A	4ステート
STA	WS	13ステート
JNZ	WAIT	7ステート
POP	PSW	10ステート
RET		10ステート

となるから、全ステート数 T は

$$T = 12 + 4 + 13 + 256 \times (13 + 4 + 13) + 255 \times 10 + 7 + 10 + 10$$

$$= 10,286 \text{ ステート}$$

となります。ここで、8085のクロックに6 MHzのクリスタルを使用したので、クロックは

$$t_{cr} = \frac{2}{6 \times 10^6} = 333 \times 10^{-9}$$

333ナノ秒となります。よってこのプログラムによって消費される時間は

$$\tau = 333 \times 10^{-9} \times 10,286 = 3.43 \times 10^{-3}$$

となって、約3.5ミリ秒ぐらいとなります。そこで、1秒の時間待ちにするためには

$$1 \div 0.00343 = 291.5$$

となるから、これを292回とすると

$$292 = 256 + 36$$

となって、次のプログラムで1秒待つこととなります。

```

**
* WAIT 1 SECOND
**

                WAIT    EQU    /200
                ORG     /212
0212  C5         W1SC   PUSH   B
0213  0600       MVI    B,0
0215  CD0002     W1S1   CALL   WAIT
0218  04         INR    B
0219  C21502     JNZ    W1S1
021C  0624       MVI    B,/24
021E  CD0002     W1S2   CALL   WAIT
0221  05         DCR    B
0222  C21E02     JNZ    W1S2
0225  C1         POP    B
0226  C9         RET
                END

```

これらの時間待ちサブルーチンを使って、／80のラッチに三角波を入力するプログラムは次のようになります。

```

**
* MOTOR TEST PROGRAM
**

W1SC  EQU    /212
0000  310002   LXI    S,/200
0003  0601     MVI    B,1      *INCREMENT
0005  DB80     MTS1   IN      /80
0007  FE7F     CPI    /7F
0009  CA1A00   JZ      MTS3     *CHANGE INC TO DEC
000C  FE80     CPI    /80
000E  CA1F00   JZ      MTS4     *CHANGE DEC TO INC
0011  80       MTS2   ADD     B
0012  D380     OUT    /80
0014  CD1202   CALL   W1SC
0017  C30500   JMP    MTS1
001A  06FF     MTS3   MVI    B,/FF  *DECREMENT
001C  C31100   JMP    MTS2
001F  0601     MTS4   MVI    B,1
0021  C31100   JMP    MTS2
                END

```

このプログラムの内容は説明するまでもないでしょう。／80のところを／81

172 第5章 機械制御のインターフェース回路とDMA

と書き換えると、もう一方のモータが動き出しています。

モータを停止するには、車輪の回転を目で見えて、三角波が横軸をクロスするところで車輪がいったん止まるので、そのときにCPUをストップすればよいのですが、TRAPプログラムの練習として、非常停止プログラムを作りました。

```

**
* TRAP JUMP
**

0024 C30001          ORG    /24
                     JMP    TRAP

**
* TRAP PROGRAM
**

W1SC EQU    /212
0100 DB80          TRAP   ORG    /100
0102 47             IN     /80
0103 DB81           MOV    B,A
0105 4F             IN     /81
0106 CD1202        TRP1   MOV    C,A
0109 78             CALL  W1SC
010A B7             MOV    A,B
010B CA1501         ORA    A
010E FA2401         JZ     TRP3
0111 3D             JM     TRP5
0112 47             DCR    A
0113 D380           TRP2   MOV    B,A
0115 79             TRP3   OUT    /80
0116 B7             MOV    A,C
0117 CA2C01         ORA    A
011A FA2801         JZ     TRP7
011D 3D             JM     TRP6
011E 4F             DCR    A
011F D381           TRP4   MOV    C,A
0121 C30601         OUT    /81
0124 3C             JMP    TRP1
0125 C31201         INR    A
0128 3C             JMP    TRP2
0129 C31E01         INR    A
012C 78             TRP7   MOV    A,B
012D B7             ORA    A
012E C20601         JNZ    TRP1
0131 76             HLT
                     END

```

8085のTRAP割込みは／24番地にコントロールが飛ぶので、ここにジャンプ命令を入れておいて、／100番地に飛びます。ここからが非常停止プログラムですが

IN ／80, IN／81

の命令で現在のラッチの情報をCPUに取り込み、それからまず、1秒間待ちます。

このところが重要であって、ラッチへのデータのセットがTRAPプログラムに入る直前に行なわれた可能性もあるので、TRAPに入ったところで必ず1秒待って、それからラッチの正負を判別して、正ならばデクリメント、負ならばインクリメントして、両方のラッチの内容がゼロになったところでHALTします。

三角波形の入力による車輪の回転テストが無事終了したので、最終性能試験として、電気自動車を直線走行させて、ラッチに送る2進数のデータと電気自動車の速度の関係を求めることにしました。

実験室の床の上に2本の直線を引いておき、電気自動車が手前の線を横切ったときにストップウォッチをスタートさせ、最後の線に達したときにストップウォッチを止めます。線間の距離を巻尺で測っておくと、速度は

距離÷時間

の形で計算できます。CPUからプログラムで台形の入力波形を入れ、電気自動車を走らせて、データを取りました。結果は図5.11のようになりました。横軸はラッチにセットする数値で、縦軸は電気自動車の速度の計測値です。

CPUからラッチに送る数値が小さいと、電気自動車は全然動きません。これは、自動車のいろいろなところに散在する摩擦力に、モータの出力が打ち勝てないためです。ラッチのデータを徐々に増していくと、データが14～20ぐらいのところで、電気自動車は動いたり動かなかったり不安定な動作をします。

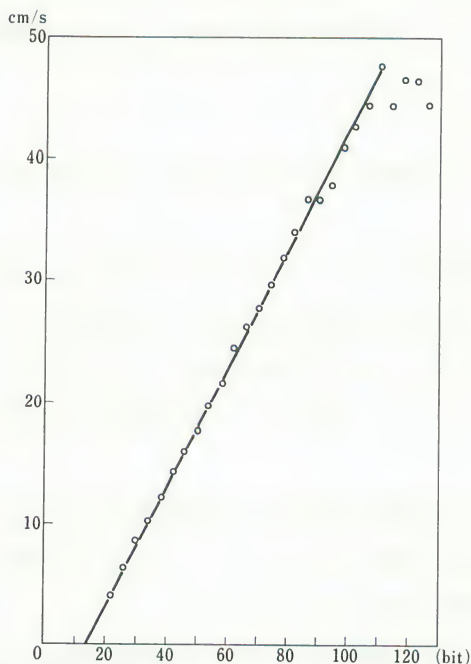


図 5.11 ラッチにセットするデータと
電気自動車の速度

たとえば、右と左のモータにまったく同じ電圧を与えても、右は動くのに左は動かない、というようなことが起こります。これは機械の軸受けとかウォーム歯車の設定が、左右まったく対称にできないため、摩擦力にわずかな差があるためと思われます。こういうところは、どんなに丁寧に製作しても、左右が完全に対称になるということはありません。多少の差異はやむを得ないところです。

このように、アンプは電圧を出力しているのに、機械は静止しているというところを不感帯とかデッドゾーンなどといいます。ここでも、約 0 ～ 14 ぐらいの不感帯があることがわかりました。

全体としてみると、電気自動車の速度はCPUの指令値にほぼ比例しています。最小2乗法という計算法を適用して、データに直線をあてはめたところ、

$$v = 0.4871b - 6.5442$$

という実験式を得ました。ここで、 b はラッチにセットするビット数です。われわれの製作した電気自動車は第一の目標を達成したものとして、次のステップに進むことにしました。

5.4 ロータリーエンコーダの インターフェース回路

機械を制御するに当たって、駆動系の代表として直流モータを取り上げ、これをマイクロコンピュータの指令によって動かすことを試みました。機械制御のもう一つの要素は、動く部分の現在位置を測定して、これをマイクロコンピュータに転送することです。

ここでは、光電式ロータリーエンコーダを取り上げ、それによって、電気自動車の車輪の回転角を計測するためのインターフェース回路について述べます。

実際に使用したロータリーエンコーダは図5.12の寸法をもつ、日本光学工業(株)のRIE-A型です。これを2個買ってきて、電気自動車の左右の車輪にフレキシブルジョイントを介して直結しました。

このロータリーエンコーダの電氣的な内容は、図5.13のようになっています。フォトダイオードとランプが1個あって、この端子がそのまま外に出てきています。ランプには+3ボルトDCを供給し、フォトダイオードには+12ボルトDCを供給します。

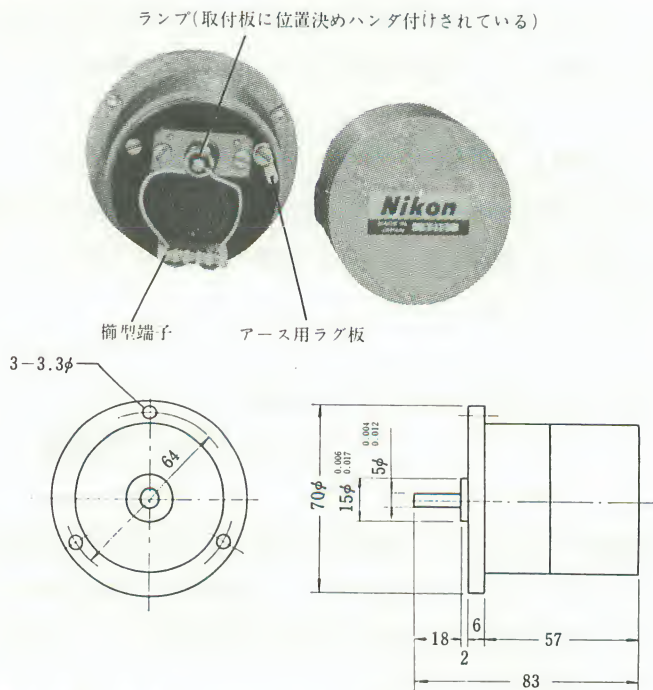


図 5.12 光電式ロータリーエンコーダの寸法(日本光学RIE-A型)

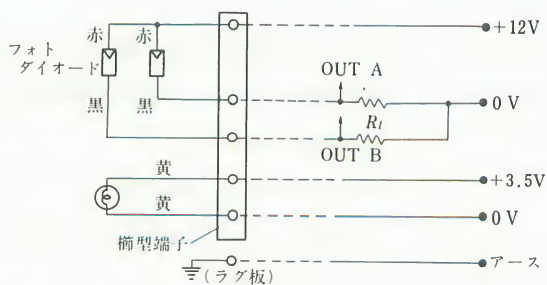


図 5.13 ロータリーエンコーダの電気的構成

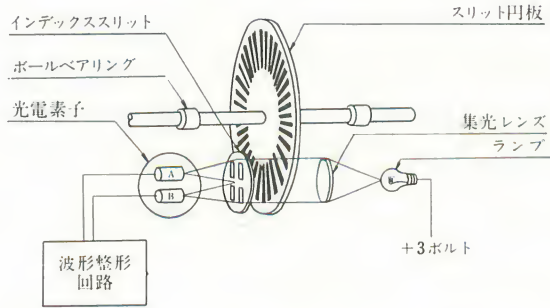


図5.14 ロータリーエンコーダの内部構造

ロータリーエンコーダの中は図5.14のようになっています。白と黒の細い縞を印刷した円板が回転するようになっています。軸が1回転すると、フォトダイオードに600の信号が発生します。これを普通600パルスのロータリー

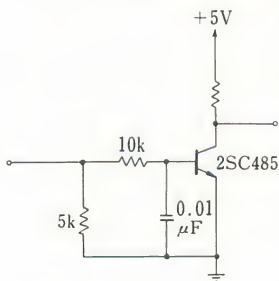


図5.15 フォトダイオードの増幅回路

エンコーダといいます。

フォトダイオードの出力電流はあまり大きくないので、図5.15の回路によって、整形増幅することにしました。これでTTLレベルの出力が得られました。

ロータリーエンコーダのなかに2つのフォト

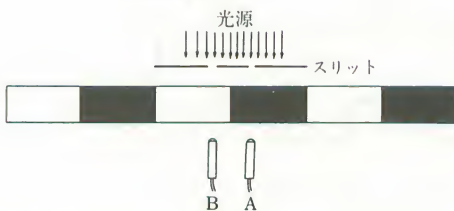


図5.16 90°位相をずらしたフォトダイオード

ダイオードが入っているのは、回転の方向を弁別するためのものですが、その原理を簡単に説明します。円板では説明がしにくいので、図5.16のように、円板を平面に展開して考えます。2つのフォトダイオードA、Bは白黒の縞模様の

変化にたいして、 90° 位相をずらして置きます。

図のような状況で、縞の帯を右の方向に引っ張ると、左側のフォトダイオードBのほうが $\frac{1}{4}$ 波長だけ早く光を受けるので、出力波形を整形したものは

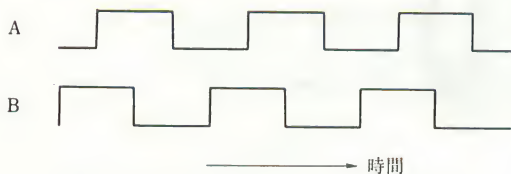


図 5.17 右に回転したときの出力波形

図5.17のようになります。

この場合、BがAよりも 90° 位相が進んでいるといえます。

逆に、図5.16の白黒の帯

を向かって左の方向に引っ張ると、今度はAのフォトダイオードが $\frac{1}{4}$ 波長早

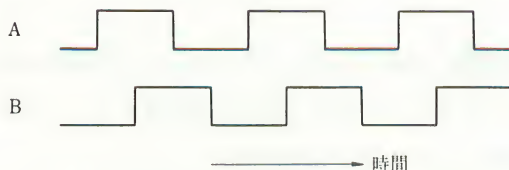


図 5.18 左に回転したときの出力波形

く光を受けるので、出力波形は当然図5.18のようになります。

この2種の波形をどこで区別するかということですが、

このためにAの波形の立上りのエッジのところに着目してみます。Aの波形が立ち上がるとき、図5.17の右回転のときは $B = 1$ であるのに、図5.18の左回転のときには $B = 0$ です。Aの波形の立下りのエッジはその逆になっていて、右回転のときは $B = 0$ で、左回転のときに $B = 1$ です。これをまとめると図5.19のようになります。

同様に、Bの波形のエッジについて考えてみると、Bの立上りのエッジは

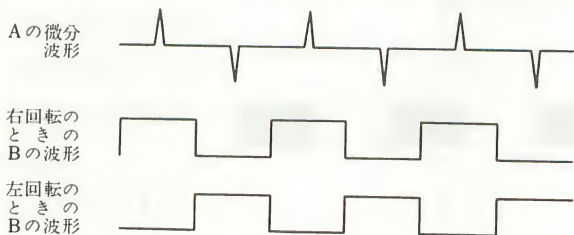


図 5.19 Aの微分波形とBの波形の関係

表 5.3

	右 回 転	左 回 転
A の立上り	B = 1	B = 0
A の立下り	B = 0	B = 1
B の立上り	A = 0	A = 1
B の立下り	A = 1	A = 0

右回転のときに $A = 0$ で、左回転のときに $A = 1$ となり、B の立下りのエッジは右回転のとき $A = 1$ で、左回転のときに $A = 0$ となります。以上をまとめると表 5.3 のようになります。ここで A の立上りパルスを $A \uparrow$ と書き、

A の立下りパルスは $A \downarrow$ と書くと B についても同じように考えて

$$A \uparrow * B + A \downarrow * \overline{B} + B \uparrow * \overline{A} + B \downarrow * A$$

が右回転の場合のパルスを構成し、逆に、

$$A \uparrow * \overline{B} + A \downarrow * B + B \uparrow * A + B \downarrow * \overline{A}$$

が左回転のパルスになります。このパルスを UP-DOWN カウンタ（たとえば SN74193）に入れると、右回転するごとにカウンタがアップカウントし、左回転するごとにダウンカウントするカウンタが作れます。

実際に製作したインターフェース回路を図 5.20 に示します。インポートの回路は省略しましたが、次のように割り当てました。

```

I N  /20   右車輪下位
I N  /21   右車輪上位
I N  /22   左車輪下位
I N  /23   左車輪上位
O U T /20  右カウンタクリア
O U T /21  左カウンタクリア

```

プログラムの例として、右車輪のカウンタをまずクリアして、1 秒後にその内容をメモリの WS 番地とその次の番地に読み込むプログラムを作ると次のようになります。

```

**
* READ RIGHT WHEEL ANGLE
**

W1SC EQU /212
      ORG /300
0300 310002 LXI S, /200
0303 D320 OUT /20
0305 CD1202 CALL W1SC
0308 DB20 IN /20

      *CLEAR COUNTER
      *WAIT 1 SEC
      *READ LOWER BYTE OF COUNTER

```


030A	321303	STA	WS	*READ HIGHER BYTE
030D	DB21	IN	/21	
030F	321403	STA	WS+1	
0312	76	HLT		
0313		DS	2	
		END		

電気自動車が直線運動をしているときには、ロータリーエンコーダの読みが、ただちに走行距離になるから、問題はありません。いま、電気自動車が直線運動をしているとして、車輪の径を R 、カウンタのパルスを N とすると、車輪が 1 回転することによって $600 \times 4 = 2400$ のパルスがくるから、電気自動車の走行距離 S は

$$S = \frac{N}{2400} \times 2\pi R$$

となります。工作機械の刃物台がベットの上面を動くようなときには、この計算で位置がわかります。

ところが、電気自動車の場合、右と左のモータに与える制御電圧が違くと、直線運動はしないで曲線運動をするので問題はかなりややこしくなります。

われわれは電気自動車のフレームにマジックペンを取り付けて、床の上に走行した軌跡を書きつけて測定しましたが、その一例を紹介すると、たとえば図5.21のようになります。これは最初に、右車輪の電圧を左車輪よりも高くしておいてスタートし、ある時間経過した後に、今度は左車輪のほうの電圧を右車輪よりも高くした結果です。予想されるように、軌跡は S 字形になります。

こういう場合に、右と左の車輪の回転角度を知って、平面上の走行軌道を計算することは不可能でしょうか。これは実は可能です。われわれの答えを以下で簡単に説明します。

まず、時刻 t のときの電気自動車の位置ははっきりわかっているものとします。このとき、ロータリーエンコーダの角度を測定しておきます。たとえば、右車輪のカウンタの読みが N_1 、左車輪のカウンタの読みが N_2 だったとし

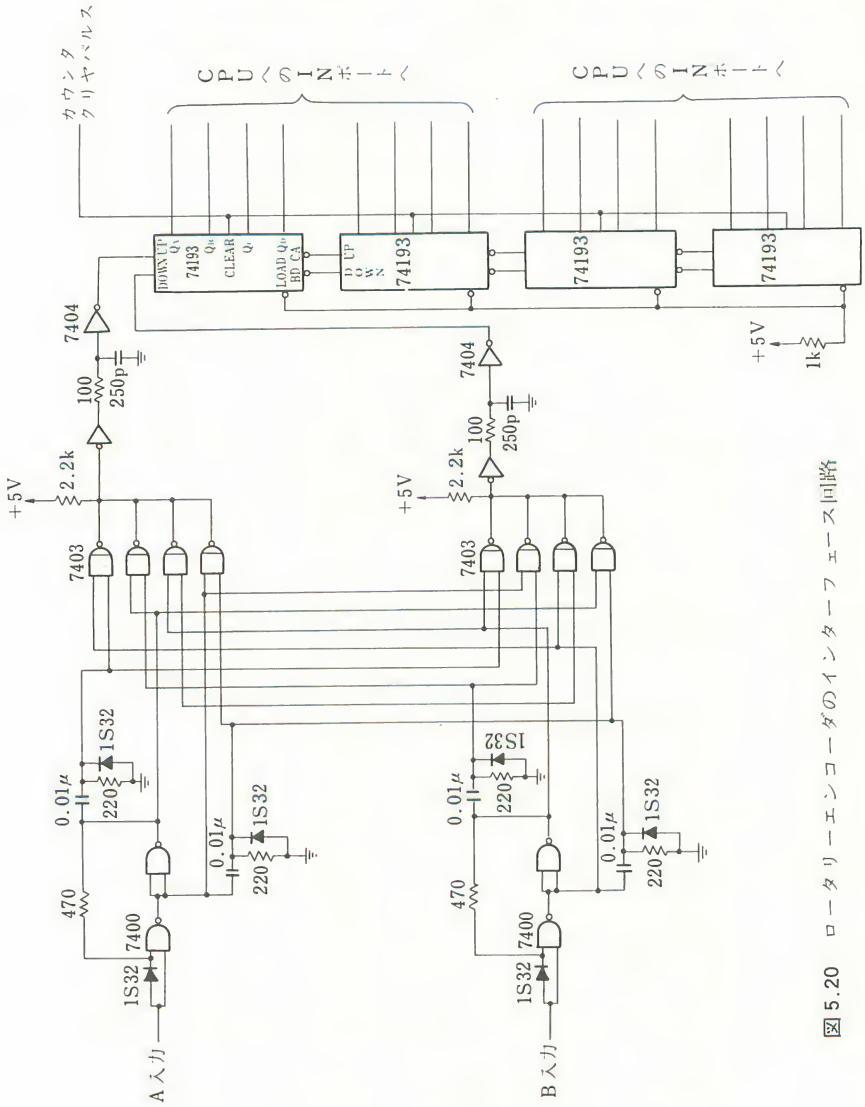


図 5.20 ロータリーエンコーダのインターフェース回路

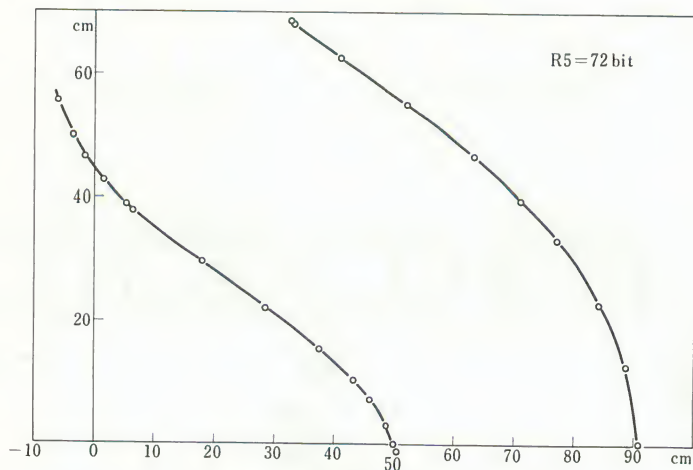


図 5.21 電気自動車の走った跡

ます。

次に、ごくわずかが時間が経過した後に、再びロータリーエンコーダの角度を測定します。時間の増分を Δt と書き、右車輪のカウンタの読みを N_1' 、左車輪の読みを N_2' とします。そうすると、 Δt というわずかな時間のなかで右車輪が走った距離 ΔS_1 は

$$\Delta S_1 = \frac{N_1' - N_1}{2400} \times 2\pi R$$

となり、左車輪は同様にして

$$\Delta S_2 = \frac{N_2' - N_2}{2400} \times 2\pi R$$

となります。そうすると Δt という時間のなかでの平均速度は右車輪について

$$v_1 = \frac{\Delta S_1}{\Delta t} = \frac{N_1' - N_1}{\Delta t} \frac{\pi R}{1200} \dots\dots\dots (1)$$

となり、左車輪について

$$v_2 = \frac{\Delta S_2}{\Delta t} = \frac{N_2' - N_2}{\Delta t} \frac{\pi R}{1200} \dots\dots\dots (2)$$

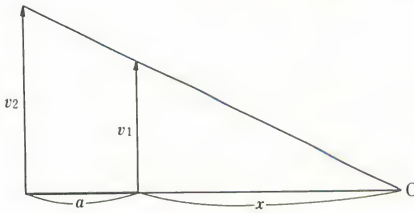


図 5.22 剛体の回転の中心

となります。さてそこで図5.22を見て下さい。簡単のために、時刻 t において、左車輪が座標の原点 $(0, 0)$ にいて、右車輪は $(a, 0)$ にいたとします。そこで、 a は電気自動車の車輪の間隔です。もし、 $v_1 \neq v_2$ なら

ば、電気自動車は図のように点 O を中心にした円運動をしているものと考えられます。 v_1 と v_2 は平行だから、三角形の相似則を使って

$$v_2 : v_1 = (a+x) : x$$

となり、これを解くと

$$x = \frac{a v_1}{v_2 - v_1}$$

となります。回転の角速度を ω と書くと、

$$\omega = \frac{v_1}{x} \text{ または } \omega = \frac{v_2}{a+x}$$

だから、これを計算すると

$$\omega = \frac{v_2 - v_1}{a}$$

となり、回転角を $\Delta\theta$ と書くと

$$\Delta\theta = \omega \Delta t$$

だから

$$\Delta\theta = \frac{v_2 - v_1}{a} \Delta t$$

となります。さて、ここで図5.23の関係から、右車輪の縦軸上方への移動距

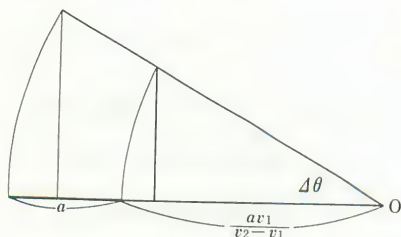


図 5.23 実際に移動した距離
移動した距離は

離は

$$\Delta y_1 = x \sin \Delta \theta = \frac{a v_1}{v_2 - v_1}$$

$$\sin \frac{v_2 - v_1}{a} \Delta t \dots \dots \dots (3)$$

となります。右車輪が横軸の右方向に

$$\Delta x_1 = x - x \cos \Delta \theta = \frac{a v_1}{v_2 - v_1} (1 - \cos \frac{v_2 - v_1}{a} \Delta t) \dots \dots \dots (4)$$

となります。左車輪についてもまったく同じだから

$$\Delta y_2 = (x + a) \sin \Delta \theta = \frac{a v_2}{v_2 - v_1} \sin \frac{v_2 - v_1}{a} \Delta t \dots \dots \dots (5)$$

$$\Delta x_2 = (x + a) (1 - \cos \theta) = \frac{a v_2}{v_2 - v_1} (1 - \cos \frac{v_2 - v_1}{a} \Delta t) \dots \dots \dots (6)$$

となります。(3)(4)(5)(6)式のなかで、変数は v_1 , v_2 , a , Δt だが、 v_1 と v_2 は (1)(2)式で計算できるので、結局変数は N_1 , N_1' , N_2 , N_2' , R , a , Δt となります。 N はカウンタの読みであるし、 Δt はカウンタを続けて読んだ時間間隔で、 R は車輪の半径、 a は車輪間隔だから、これは全部測定可能な量です。だから、電気自動車が行走している間に、右と左の車輪のロータリーエンコーダの値を読んでいくと、電気自動車が走った軌道が時々刻々に計算できることになります。

前に出てきた図5.21のなかで、実線が実際に電気自動車が走行した軌道で、白丸点がマイクロコンピュータが計算した座標です。両者はよく一致していることがわかります。図5.24の場合のように、最終段階で計算値がずれたこともあります。われわれは、現在、計算値が実際の値からずれる原因を追究していますが、残念ながら、完全に解明されたとはいえません。

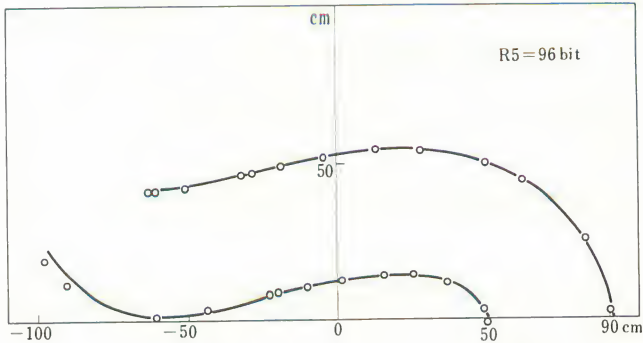


図 5.24 計算値が実際の軌道よりずれた場合

5.5 DMAによる情報の転送

これまで8080 Aのマイクロコンピュータシステムと8085のシステムについて述べてきました。最終に、これら両システムをDMAでリンクする方法について述べます。

これまで説明してきたように、8080 Aのシステムにはタイプライタをつけて、会話型モニタ、エディタ、アセンブラなどが使えるようになっています。

8085のマイクロコンピュータシステムを設計するに当たって、最初に、入出力装置をどうするかについて熟考しました。簡単にいえば、こちらにも独立の入出力タイプライタをつければよいのですが、そんなことをしたら、タイプライタがいくつあっても足りないということになってしまいます。電気自動車の上に、タイプライタを常時乗せておくことはできないので、すくなくとも電気自動車が走り出すときはタイプライタをはずさなければなりません。このように、どのみちソケットを入れたり抜いたりするのであれば、それはタイプライタのコネクタであろうと、その他のケーブルであろうと同じ

ことです。

こう考えて、電気自動車の8085のマイクロコンピュータシステムを入出力の完備した8080Aシステムのバスに直接ぶら下げることにしました。こうすることによって、

- (1) 1つの入出力装置群が2つのマイクロコンピュータシステムに共用できる。
- (2) 8080Aのシステムでメモリが不足したときに、8085のメモリを外部メモリとして利用できる。

という2つのメリットを得ました。デメリットは、勿論、8080Aの入出力装置が混雑するという点です。これは当然のことでしょう。混雑がはげしくなると、物理的に2つのシステムの入出力をさばききれなくなったら、それは当然、入出力装置を別にしなければならないということです。

こうして、8080Aを親にして、8085を子としてシステムを組みました。両システムの間には、約5メートルぐらいのケーブルを引かなければなりません。が、これにたいしては第3章で述べた回路を使いました。

8080Aのマイクロコンピュータシステム側のインターフェースですが、これは図5.25のように設計しました。

8085側の受けのインターフェース回路は図5.26のとおりです。こうすると、8080Aに主導権をもたせた形式で、情報の転送が行なわれることになります。簡単にいえば、8080A側から、8085のメモリにプログラムをロードすること、および8085のメモリのなかにある実験データを8080Aが引き取って、プリントするなり、計算処理した後に印刷するという操作が可能になります。

さて、そこで8080Aシステムのプログラムを作っていきます。

8085のマイクロコンピュータシステムがHOLD状態になっているかどうかを8080Aに読み込んで、それで8080Aのフラグをセットするサブルーチンを作ります。

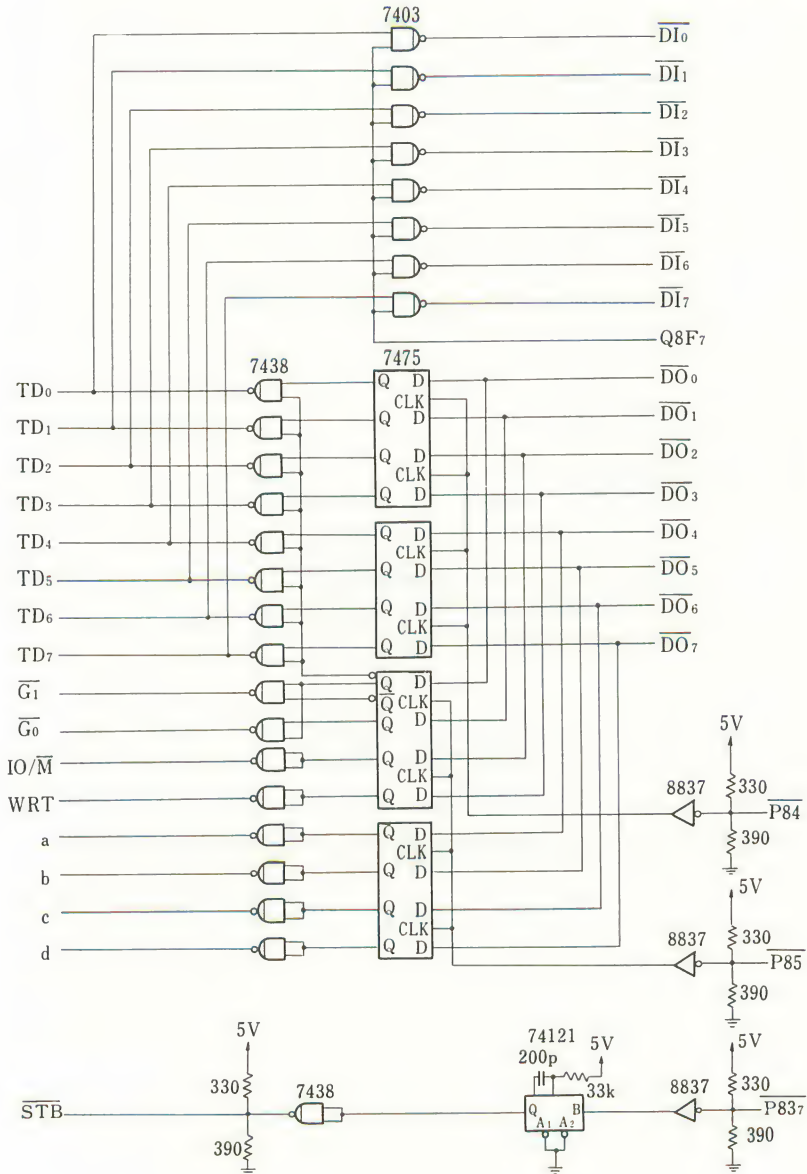
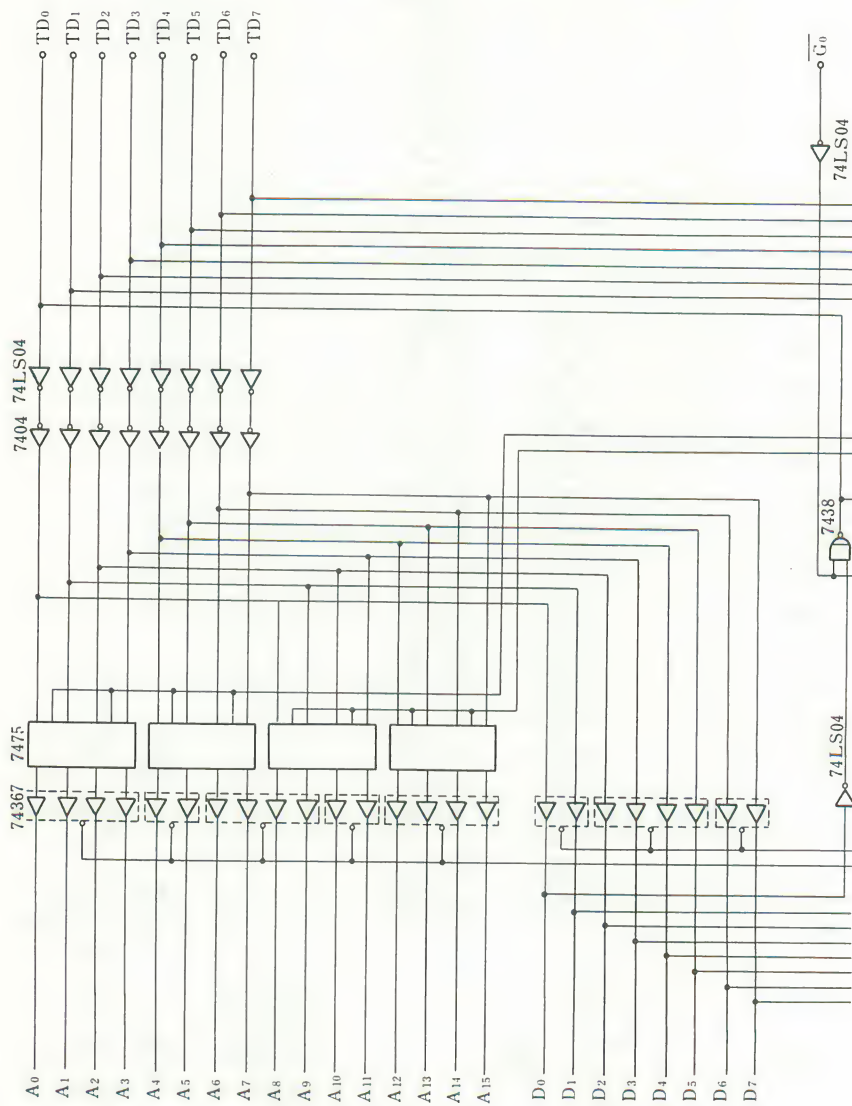


図 5.25 8080A側のインターフェース回路



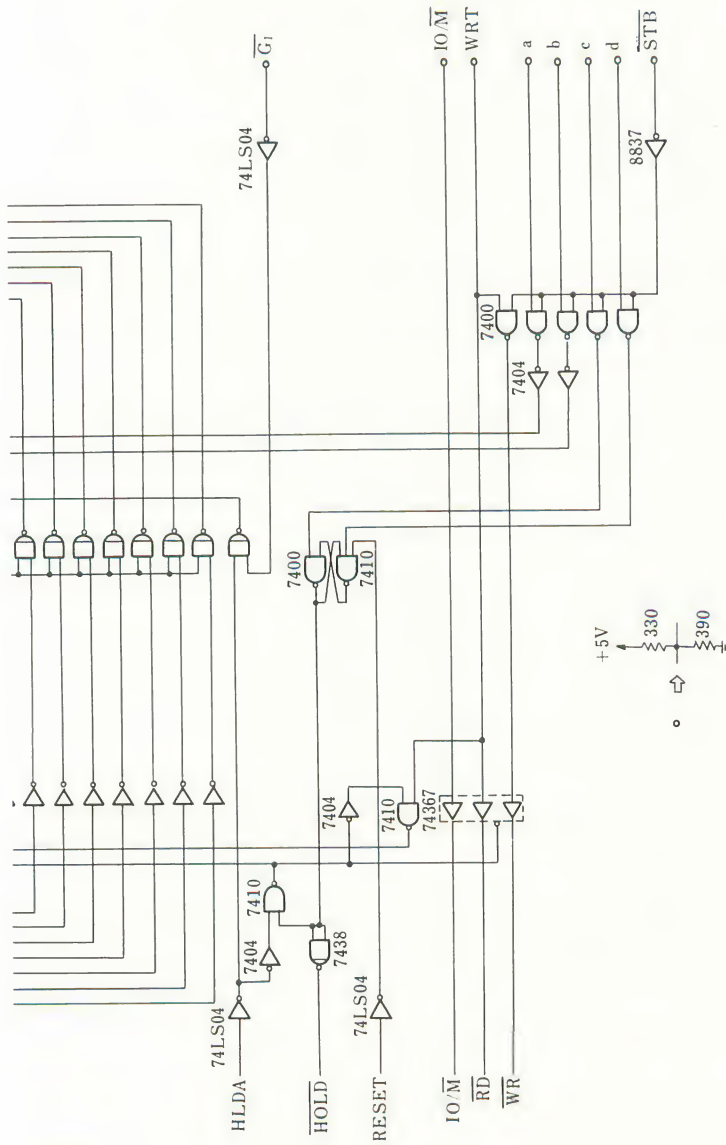


図 5.26 8085DMA インターフェイス回路


```

**
* READ 8085STATUS AND SET FLAG
**

```

```

F00D 3E02      RUST  ORG    /F00D
F00F D385      MVI    A,2
F011 3E80      OUT    /85
F013 D38F      MVI    A,/80
F015 D380      OUT    /8F
F017 D360      IN     /30
F019 E601      ANI    1
          RET
          END

```

8085がHOLD状態のときに1，HOLD状態でないときに0だから，それによって8080AのPSWのフラグがセットされます。

8080Aが8085のCPUにたいして，HOLD要求を出すフリップフロップがありますが，これをオンオフするサブルーチンを作ります。

```

**
* HOLD REQUEST FF ON/OFF SUBROUTINE
**

```

```

F01A F5      HOFPP  ORG    /F01A
F01B 3E30      PUSH   PSW
F01D C323F0    JMP     A,/80
F020 F5      HONPP  PUSH   PSW
F021 3E40      MVI    A,/40
F023 D385      OUT    /85
F025 3E80      MVI    A,/80
F027 D383      OUT    /83
F029 F1      FOP    PSW
F02A C9      RET
          END

```

HOFPPから入るとHOLD要求フリップフロップがオフとなり，HONPPから入るとオンになります。

今回製作した8085システムは，フロントパネルがHOLD要求を出す可能性をもっているのです，8080Aからの要求がそれとから合わないようになければなりません。

2つのHOLD要求をさばく方法ですが，要求に優先度をつけて，優先度の高い方が要求を出しているときは，優先度の低い方の要求をハード的に押さえてしまうようにするのが普通ですが，ちょっと回路がめんどくさくなるので，ここでは優先度はつけないで，ソフトウェアで2つのHOLD要求が

同時に出ないようにしました。

8080 Aが8085のCPUにホールドをかけて、情報の転送ができるように前準備するサブルーチンにOPENという名前をつけました。その手順のフローチャートは図5.27のようになります

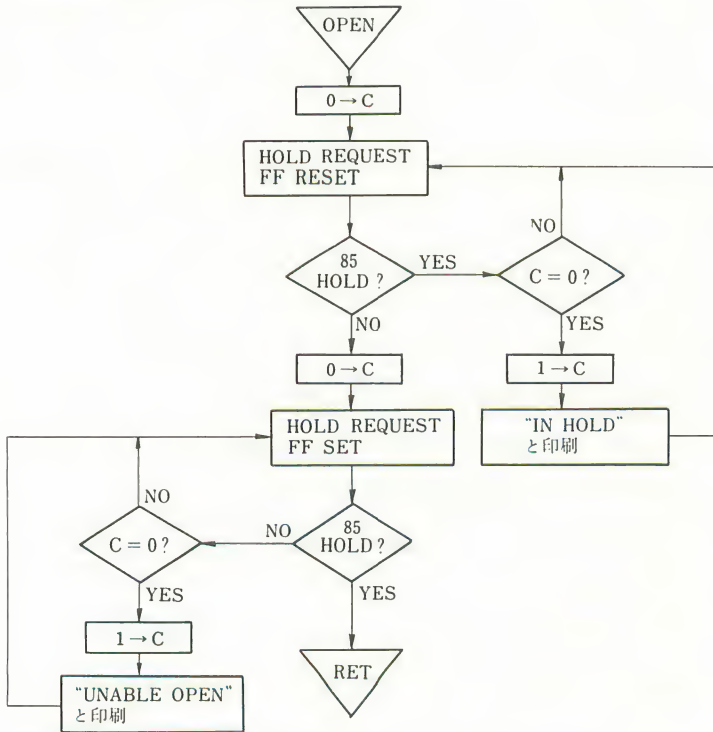


図 5.27 OPENルーチンのフローチャート

最初に、必ず、HOLD要求フリップフロップをリセットします(たいていの場合、このフリップフロップはオフになっているのですが、念のため行ないます)。そこで、8085の状態を読み込んでHOLD状態かそうでないかを調べます。こちらのHOLD要求をオフにしているのだから、これで8085がHOLD状態ならば、それは別のソースからHOLD要求が出ているに違いありません

ん。

このときには、タイプライタに

IN HOLD

と印刷して、オペレーターにたいして、8085のHOLD要求源を調べるように注意します。手順としては、このループを繰り返して、8085のHOLD要求がすべて解除するまで待ちます。ただし、印刷は一度だけでよいから、続けて印刷はしません。

8085のHOLD状態が解けると、コントロールが次に移り、今度は8080AからHOLD要求フリップフロップをオンにします。この要求は8085によって受け付けられるか、受け付けられないかのいずれかです。もし、受け付けられれば、8085はHOLD状態になったわけだから、このOPENサブルーチンの目的は達せられたわけで、これでメインルーチンに帰ります。

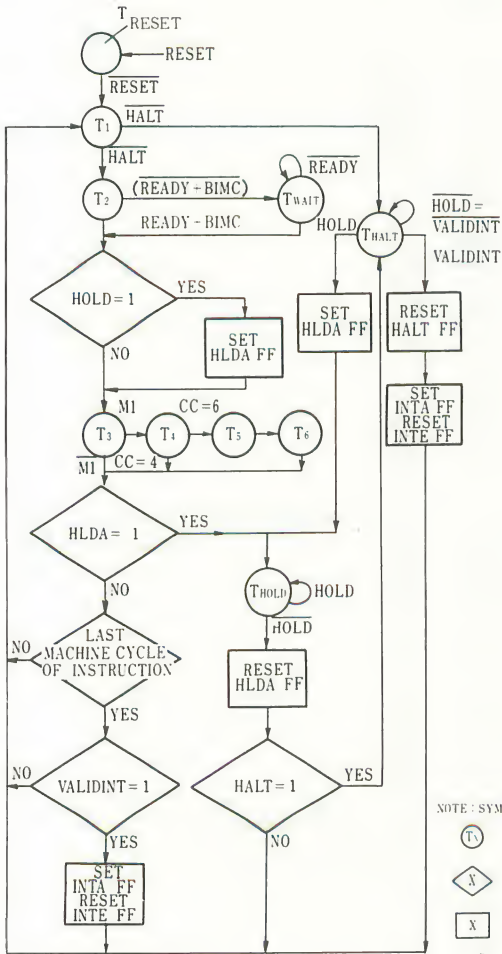
もし、8080AからHOLD要求を出して、それが8085によって受け付けられないとすると、唯一の可能性は、8085がWAIT状態になっているときです。

図5.28は8085のCPUの状態遷移図ですが、これから見てもわかるように、CPUがWAIT状態に入ると、そのWAITが解けるまで、CPUは他の一切の要求を受け付けません。

CPUがHOLD状態にならなかつたら、タイプライタから

UNABLE OPEN

と印刷して、このループを続けます。このとき、オペレーターは8085のWAIT状態を解かなければなりません。OPENサブルーチンのプログラムは次のようになります。



NOTE : SYMBOL DEFINITION



= CPU STATE T, ALL CPU STATE TRANSITIONS OCCUR ON THE FALLING EDGE OF CLK.



= A DECISION (X) THAT DETERMINES WHICH OF SEVERAL ALTERNATIVE PATHS TO FOLLOW.



= PERFORM THE ACTION X.



= FLOWLINE THAT INDICATES THE SEQUENCE OF EVENTS.



= FLOWLINE THAT INDICATES THE SEQUENCE OF EVENTS IF CONDITION X IS TRUE.

CC

= NUMBER OF CLOCK CYCLES IN THE CURRENT MACHINE CYCLE.

BIMC

= "BUS IDLE MACHINE CYCLE" = MACHINE CYCLE WHICH DOESN'T THE SYSTEM BUS.

VALIDINT = "VALID INTERRUPT" =

TRAP + INTE + INTR + RSTA + \overline{MA} + RSTB + \overline{MB} + RSTC + \overline{MC}

図 5.28 8085CPUの状態遷移図

**

* SUBROUTINE OPEN

* HOLD 80R5

**

		HOPP	EQU	/F01A
		RDST	EQU	/F00D
		PRSP	EQU	/40D9
		HONP	EQU	/F02C
			ORG	/F02B
F02B	F5	OPEN	PUSH	PSW
F02C	C5		PUSH	B
F02D	E5		PUSH	H
F02E	0E00		MVI	C,0
F030	CD1AF0	OPN1	CALL	HOPP
F033	CD0DF0		CALL	RDST
F036	CA56F0		JZ	OPN2
F039	79		MOV	A,C
F03A	B7		ORA	A
F03B	C230F0		JNZ	OPN1
F03E	0C		INR	C
F03F	3E02		MVI	A,2
F041	CDD940		CALL	PRSP
F044	3E07		MVI	A,7
F046	214FF0		LXI	H,OPM1
F049	CD00F0		CALL	PSTR
F04C	C330F0		JMP	OPN1
F04F	79451068464364			
		OPM1	DB	/79,/45,/10,/68,/46,/43,/64
F056	4F	OPN2	MOV	C,A
F057	CD20F0	OPN3	CALL	HONP
F05A	CD0DF0		CALL	RDST
F05D	C281F0		JNZ	OPNF
F060	79		MOV	A,C
F061	B7		ORA	A
F062	C257F0		JNZ	OPN3
F065	0C		INR	C
F066	3E02		MVI	A,2
F068	CDD940		CALL	PRSP
F06B	3E0B		MVI	A,/B
F06D	2176F0		LXI	H,OPM2
F070	CD00F0		CALL	PSTR
F073	C357F0		JMP	OPN3
F076	344561624375			
		OPM2	DB	/34,/45,/61,/62,/43,/75
F07C	1046577545			
			DB	/10,/46,/57,/75,/45
F081	AF	OPNF	XF/,	A
F082	D385		OUT	/85
F084	E1		POP	H
F085	C1		POP	B
F086	F1		POP	PSW
F087	C9		RET	


```

**
* PRINT STRING
**

PRNT EQU /4058
ORG /F000
PSTR PUSH B
F001 47 MOV B,A
F002 7E MOV A,M
F003 CD5840 CALL PRNT
F006 23 INX H
F007 05 DCR R
F008 C202F0 JNZ PST1
F00B C1 POP B
F00C C9 RET
END

```

*PRINT SUB IN MONITOR

次に、すべての処置が終わった後に、HOLD要求をオフにする手順ですが、このサブルーチンにCLOSEという名前をつけました。フローチャートは図5.29です。HOLD要求フリップフロップをオフにし、HOLD状態が解けたことを確認してメインに戻ります。

もし、HOLDをオフにしてもHOLD状態が解けないとすると、こういうこ

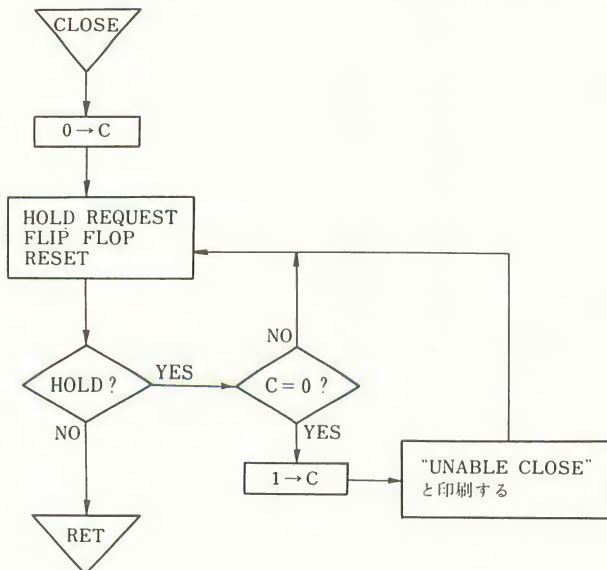


図 5.29 サブルーチンCLOSEのフローチャート

とは他のHOLD要求が無いことを確認しているのだから論理的には起こり得ないのだが、誤動作などでも起こってこういう事態になったのかもしれないが、このときにはタイプライタから

UNABLE CLOSE

と印刷します。このときオペレーターは8080A以外のどこからHOLD要求が出たか調べる必要があります。CLOSEサブルーチンのリストは次のようになります。

```

**
* SUBROUTINE CLOSE
**

      HOFB EQU /F01A
      RDST EQU /F00D
      OPNF EQU /F081
      PRSP EQU /40D9
      PSTR EQU /F000
      OFG EQU /F088
F088 F5 CLSE PUSH PSW
F089 C5 PUSH B
F08A E5 PUSH H
F08B 0E00 MVI C,0
F08D CD1AF0 CLS1 CALL HOFB
F090 CD0DF0 CALL RDST
F093 CA81F0 JZ OPNF
F096 79 MOV A,C
F097 B7 ORA A
F098 C28DF0 JNZ CLS1
F09B 0C INR C
F09C 3E02 MVI A,2
F09E CDD940 CALL PRSP
FOA1 3E0C MVI A,/C
FOA3 21ACF0 LXI H,CLSM
FOA6 CD00F0 CALL PSTR
FOA9 C38DF0 JMP CLS1
FOAC 344561624375 CLSM DB /34,/45,/61,/62,/43,/75
F0B2 107343463275 DB /10,/73,/43,/46,/32,/75
      END

```

*PRINT SPACE IN MONITOR

次に、8085Aから8085のDMAインターフェースカードのアドレスラッチにアドレス情報をセットするサブルーチンを作ります。リストは次のようになります。

```

**
* SEND ADDRESS
**

FOB8 F5      ADRS  ORG    /FOB8
FOB9 7D      PUSH  PSW
FOBA D384    MOV    A,L
FOBC 3E11    OUT    /84
FOBE D385    MVI    A,/11
FOC0 3E80    OUT    /85
FOC2 D383    MVI    A,/80
FOC4 7C      OUT    /83
FOC5 D384    MOV    A,H
FOC7 3E21    OUT    /84
FOC9 D385    MVI    A,/21
FOCB 3E80    OUT    /85
FOCD D383    MVI    A,/80
FOCF F1      OUT    /83
FOD0 C9      POP    PSW
                RET
                END

```

HLレジスタにもったアドレスをインターフェースカードに送ります。これは一本道です。

次に、8085のメモリの内容を読み出すサブルーチンですが、これは次のようになります。

```

**
* READ SUBROUTINE
**

FOD1 CDB8F0  ADRS  EQU    /FOB8
FOD4 AF      REDV  ORG    /FOD1
FOD5 D385    CALL  ADRS
FOD7 3E80    XRA    A
FOD9 D38F    OUT    /85
FODB DB80    MVI    A,/80
FODD C9      OUT    /8F
                IN     /80
                RET
                END

```

HLレジスタにアドレスをもち、このサブルーチンに飛ぶと、Aレジスタに8085のメモリの内容をもって帰ってきます。

8080 Aから8085のメモリに情報を書き込むサブルーチンは次のようになります。

```

**
* WRITE SUBROUTINE
**

      ADRS    EQU    /F0B8
      FODE    CDB8F0  WEDV  ORG    /F0DE
      F0E1    D384    CALL  ADRS
      F0E3    3E09    OUT   /84
      F0E5    D385    MVI   A,9
      F0E7    3E80    OUT   /85
      F0E9    D383    MVI   A,/80
      F0EB    C9      OUT   /83
                        RET
                        END

```

HLレジスタにアドレスをもち、Aレジスタにデータをもって、このサブルーチンに飛ぶと、そのデータが8085のメモリに書き込まれます。

8080 Aから8085のメモリにアクセスするには、これまで説明してきたように、情報の転送を行なう前後に、OPENとCLOSEのサブルーチンと呼ばなければなりません。データのブロックを転送する場合には、最初にOPENして、情報の転送を行ない、最後にCLOSEするというシーケンスになります。

別に、1語を転送する前後で、OPENとCLOSEを毎行行なう手順も考えられます。これをスタンドアロン型の情報転送といいます。スタンドアロン型サブルーチンはすべての手順がそこだけでつくされているので、能率の悪い点はあるかもしれませんが、気楽に使えるというメリットがあります。スタンドアロン型のREADとWRITEのサブルーチンは次のようになります。

```

**
* STAND ALONE READ/WRITE
**

      OPEN    EQU    /F02B
      REDV    EQU    /F0D1
      WEDV    EQU    /F0DE
      CLSE    EQU    /F088
                        ORG    /F0EC
      F0EC    CD2BF0  RD85  CALL  OPEN
      F0EF    CDD1F0  CALL  REDV
      F0F2    CD88F0  RWFN  CALL  CLSE
      F0F5    C9      RET
      F0F6    CD2BF0  WR85  CALL  OPEN
      F0F9    CDDEF0  CALL  WEDV
      F0FC    C3F2F0  JMP    RWFN
                        END

```

このシステムでは、情報の転送は多くの場合、ブロック転送になるので、

ブロック転送のサブルーチンを次のように作りました。

```

**
* MOV 80 TO 85
**

F100 0A      WR85 EQU /F0F6
F101 CDF6F0  MTOV ORG /F100
F104 03      MTOV LDAX B
F105 23      MTOV CALL WR85
F106 79      MTOV INX B
F107 BB      MTOV INX H
F108 C200F1  MTOV MOV A,C
F10B 78      MTOV CMP E
F10C BA      MTOV JNZ MTOV
F10D C200F1  MTOV MOV A,B
F110 C9      MTOV CMP D
          MTOV JNZ MTOV
          RET

**
* MOVE 80 FROM 85
**

F111 CDECF0  RD35 EQU /F0EC
F114 02      MFRV CALL RD35
F115 03      MFRV STAX B
F116 23      MFRV INX B
F117 7B      MFRV INX H
F118 BD      MFRV MOV A,E
F119 C211F1  MFRV CMP L
F11C 7A      MFRV JNZ MFRV
F11D BC      MFRV MOV A,D
F11E C211F1  MFRV CMP H
F121 C9      MFRV JNZ MFRV
          RET
          END

```

8080 Aから8085に転送する場合、BCレジスタに8080 Aのほうのブロックのスタートアドレス、DEに8080 Aのブロックのエンドアドレス+1を置き、HLレジスタに8085のメモリのスタートアドレスを置きます。これでMTOVに飛ばせば、8080 Aのメモリの内容が8085のメモリに転送されます。

逆に、8085のメモリの内容を8080 Aに転送するには、BCレジスタに8080 Aのブロックのスタートアドレスを置き、HLレジスタに8085のブロックのスタートアドレスを置き、DEレジスタに8085のエンドアドレス+1を置きます。これでMFRVに飛ばせば、目的とする情報の転送ができます。転送する情報の内容は目的によって違うので皆さんで考えてみて下さい。

5.6 ノイズと誤動作

これまで、いくつかのインターフェース回路について述べてきましたが、その最後に、電子回路とノイズの問題について、私の考えを述べておきたいと思います。

ディジタル回路の原理はまことに簡単なものですが、実際に、それらを組み立てて動かしてみると、思った通りに動かないということがよくあります。とくに、私のように、初心者を指導するという立場にいと、こういう誤動作の問題にぶつかることが多くあります。

しかし、ここで私がはっきりといいたいことは、ノイズとか誤動作といっても、それらのほとんどが電子技術の基礎を十分に理解していないことから起こるのであって、本当にノイズと呼ぶようなものによる誤動作には、そうめったにあるものではないということです。

これまでに経験したいろいろのケースのなかで、どう対策を立てたらよいかまったくわからなかったという誤動作の事例はほとんどありません。確実な動作を得るために、私が採用している原則は、

- (1) 部分ごとに検査できるような設計を行なうこと
- (2) 過去において成功した設計はよほどの理由がない限り変更しないこと
- (3) 誤動作が発生したら、必ず、原因を定量的に追究すること

などです。原因が定量的に追究できない誤動作が起こると、私の考えでは、これが本当のノイズによる誤動作になるのではないかと思います。こういう誤動作は、めったに起こらないものであって、しかも原因を論理的につきとめられないのだから、どうすることもできません。これは、おそらく使用する素子の品質を高度化しないと解決できない問題なのではないかと思えます。

これまでのインターフェース回路の作り方を見て、読者のなかに、何故こんなばかばかしいデータの伝送をするのか不思議に思った人が多くいると思います。

ある時に、卒業研究の学生が私のところに来て、私の設計した伝送回路データの伝送率を10倍あげて500 kHzにすることができる、といってきました。そこで私は、その通りに伝送回路を変更させてみました。結果はどうだったでしょうか。データが正確に伝送される割合が約80%で、残りの20%に誤ったデータが伝送されてしまうのです。これではデジタル情報の伝送回路として落第です。

私の経験からいいますと、デジタル情報の伝送速度を多少上げるといふようなことは、たいして重要な問題ではありません。それよりも、100%確実な情報伝送を行なうほうが、どれくらい重要かわかりません。とにかく、確実な情報伝送ができることが絶対の条件です。この条件を維持したうえで何とかするのであればよいのですが、現実はそのあまくなくて、良かれと思ってした改良が、時に誤った結果を招くこともあり得るのです。

それから、1つのパルスでいくつもの仕事を同時に処理することも、できるかぎり避けるようにしています。たとえば、フリップフロップがA、B、Cと3つあったとして、これらをオンにしたいとします。時間的に制限がなければ、1つのパルスで3つのフリップフロップを同時にオンにすることもできますが、私の方針としては、こういうときにはでき得るかぎり、3種類のパルスを使って、Aをオンにし、Bをオンにし、Cをオンにするという方法を採用します。

そうすれば、プログラムが長くなるではないかということも知れませんが、これはわざといくつかの手順を並べてプログラムを長くしているのであって、CPUがプログラムのミスで暴走したようなときに、データの部分をプログラムと解釈して実行したりするときに、1つのパルスならばたまたま誤って

発生することがあるが、3つの違ったパルスが続けて出る確率は非常に小さいからです。

こういうところは、機械の制御をするときに必要となるテクニックです。わざとプログラムを冗長にして、重大な誤動作を避けるようにします。

最後に、やはりこういった実践的な問題においては常にそうですが、経験という因子が重要な地位を占めています。これについては、改めていう必要もないでしょう。

あ と が き

インターフェース回路について述べるに当たって、インターフェースという意味があまりはっきりしていないのに困りました。インターフェース技術を小さく見れば、それはマイクロコンピュータにいろいろスタンダードな入出力装置をつけるテクニックということになるでしょう。こういった、いわば職人的なテクニックの集まりとしてインターフェース技術を見ることは、私はあまり賛成できません。

たとえば、マイクロコンピュータにフロッピーディスクをつける方法とか、CRTをつける方法などは、確かにインターフェース回路の作り方の練習としてはよいでしょうが、あまり実用的な意味はありません。

勿論、私の研究室ではそういった装置を使用していますが、それらの回路の作り方を理解するのは、それを製作して商売するというような立場の人しか必要ないのであって、われわれ一般のユーザーにとって、そういった標準的な回路は、目的に応じて既製品を購入したほうが、コスト的に安上がりになります。

これからの一つの方向として、フロッピーディスクの機械のなかに、LSIのコントローラが組み込まれて、それでマイクロコンピュータのバスに簡単につなげる、というような商品がどんどん市場に出てくるものと思われます。カセットMTにしても、CRTディスプレイにしても同じことです。

そういう状況になることを前提にすると、インターフェース回路の設計技術にとって重要なことは、ユーザー各自がもっている問題をどのようにしてマイクロコンピュータに接続して、それを解決していくかということです。

私は、インターフェース技術とは、結局“マイクロコンピュータという素子を現実の世界のなかで問題解決に応用する技術”と定義します。

最近のマイクロコンピュータにたいする一般の興味の盛り上がりは、目を見はらせるものがあります。これはもう完全にブームといってよいでしょう。それはそれでまことによいのですが、ともすると、ブームに乗りすぎたような現象があらこちらで出はじめているのは、まことに残念なことです。

マイクロコンピュータが話題になるのは結構ですが、話題だけにしかないものだったら、ちょうど、あのスーパーカーのブームのように、一時のはやり物の一つになってしまいます。ある日突然に誰もいなくなってしまって誰もマイクロコンピュータに見向きもしなくなる、そういう日がやって来るに違いありません。

マイクロコンピュータはそんなつまらないものではない、と私は確信しています。

しかし、マイクロコンピュータのむつかしいところは、マイクロコンピュータの価値がマイクロプロセッサのLSIのチップのなかにあるのではなくて、マイクロコンピュータをどう使っていくかということにあること、換言すれば、マイクロコンピュータのチップのなかではなくてわれわれの頭脳のなかにある、われわれ人間がそれをどう使っていくかというその知恵のなかにあるという点です。

だから、マイクロコンピュータが真に有用な、しかもすばらしいものだということを実証するには、マイクロコンピュータのサイクルタイムを速くしたり、メモリの容量を増加したりするのではなくて、マイクロコンピュータで何ができるかを具体的に提示することが必要なのです。

こういう意味で、私はこれまでマイクロコンピュータの応用技術を普及することに全力を投入してきました。計測自動制御学会の部会の幹事をしているときに、マイクロコンピュータのアプリケーションのシンポジウムをほとんど独力で開催してきました。

こういった会を主催して感ずることは、マイクロコンピュータのブームは

華やかだが、その応用の事例はまだまことに貧弱な状態になっているということです。ブームだけが先走ってしまって、実質がそれについていけない状態になっています。

これではいけないのであって、もっともっとマイクロコンピュータのアプリケーションの分野を切り開いていかなければなりません。

しかし、ただアプリケーションが重要だと口で言っているだけではいけません。もし応用技術が重要だとすれば、それはやはり自分自身の行動のなかでそれを実践することです。

こういうことで、私は自己の全時間をマイクロコンピュータの応用技術の開発に投入してきました。CPU のチップを買ってきて、ハンダゴテを振りまわして CPU カードを作ります。なにもそんなことまでしないでも、メーカーの作ったボードを買ったらどうですか、とすすめてくれる人もいますが、私はあえて自作することにしています。そうすることによって、よりマイクロコンピュータの中味と接触できるような感じが得られるからです。

こうして、教育の現場で陣頭指揮していると、いろいろなケースにぶつかります。

あるときに、私の研究室に他校で電気工学を教えている先生が6ヵ月内地留学ということで勉強に来ました。最初に、マイクロコンピュータのキットを買ってそれを組立てました。これは確かに動きました。次に、カセットテープのインターフェース回路を作るということで、IC を買って来て組立てました。ところが、どうしても動作しないのです。

いったいどうしたのか、見てくれというので、シンクロスコープでいろいろ信号を調べてみました。おかしいことに、信号がふらふらと泳いでいます。こんなことはないのだが、と思って電源ピンにさわってみました。電源がきていないのです。よく見ると配線がしてありません。

どうしたのか聞いてみたところ、「IC というのは、電源を配線しなければ

いけないのですか」 という答えが返ってきたので、私はびっくりしてしまいました。これが専門が文科の人ならばとにかく、他人に電気工学を教える立場の人が、IC を動作させるのに、電源を供給しなければならないことを知らないとは、……私はあ然としてしまいました。

何はともかく、マイクロコンピュータの勉強をしようと思ったら、書店に行ってデジタル回路について書いてある本を二、三冊買って読む、せめてそのくらいのことはすべきでしょう。書物は安くて確実な情報を得るのにもっとも適した手段です。そういう努力なしで、まるで頭からコンクリート壁に突っ込むようなことを多くの人がやっているのです。これでは駄目です。

それから、もう一つは数学です。マイクロコンピュータを応用するためには、数学の知識が必要になります。これは何も大学などで教える高等数学の知識が必要だ、というわけではありませんが、事物を緻密に分析し、把握構成する力が絶対に必要になります。

ある時に、マイクロコンピュータの掛け算と割り算がどうしても理解できない人にぶつかったことがあります。その人は、普通の10進数の乗除算はよく知っているのです。

10進数の乗除算も2進数の乗除算も考え方はまったく同じです。こういうところが数学のきれいなところですよ。2進数の場合のほうが、構造が単純化されているから、2進数の乗除算のほうが理解しやすい場合もあります。

だから、10進数の乗除算はできるのに、2進数の乗除算は理解できないという人は、実は10進数の乗除算をただテクニックとして知っているだけで、数学として乗除算のメカニズムを理解してはいない人だ、ということになります。もっと、一つの事物について、論理的に思考をする習慣をつけておく必要があるでしょう。

このように、未熟な人の例をあげて不満を述べるのは簡単ですが、問題はそういう裾野にいる人も含めて、マイクロコンピュータの基礎技術をどのよ

うに普及していくかです。このためには、とくにコストの安い書物の形式でマイクロコンピュータの応用事例をふんだんに書いたものを出版していかなければならないと思います。

そういう意味で、ここでマイクロコンピュータのインターフェース回路の作り方について、私の経験を述べました。実際に筆をとって、はじめてわかったことは、思ったよりもインターフェース回路の説明に紙数がかかるということです。最初の計画のときには、もっとあれもこれも考えていたのですが、進行にしたがってあれもこれも削除しなければならない、ということになってしまいました。

それを補う意味もかねて、これまで私が読んだ書物とか文献のなかで、紹介する価値のあるものを選んで列記し、読者がさらに勉強するときの資料の一助にしたいと思います。

まず、エレクトロニクス回路の製作記事として、著者の息づかいが聞こえてくるような本として、

(1) 前田，電子オモチャの作り方，日本放送出版協会，1976
があります。これは私の好きな本です。ここでの主題と直接関係がないかもしれませんが、一読を薦めます。

コンピュータのハードウェアの設計についてですが、これには、

(2) Chu, Digital Computer Design Fundamentals, McGraw-Hill Book Company, 1962

(3) フィスター，デジタル計算機の論理設計，朝倉書店，1960

(4) 大川，マイクロプログラミング，培風館，1972

があります。マイクロコンピュータの設計にかんし、CPUの中味の設計法は直接必要はありませんが、知っておいて悪いはずはありません。とくに、(2)は固定小数点2進数の計算法を詳しく述べていますし、(4)は浮動小数点数の計算法について詳しく述べています。そういうアルゴリズムの習得にも役

立ちます。

ディジタル回路についてですが、これは現在のところ、メーカーの発表している資料がもっとも役立つようです。とくに、TI のエンジニアが執筆した

- (5) Morris 他, Designing with TTL Integrated Circuits,
McGraw-Hill Book Company, 1971

は読んでおきたい書物の一つです。この和訳として、

- (6) 梅原他訳, TTL 特性の応用, エレクトロニクス ダイジェスト,
1974

がありますが、とても読みにくい文章になっています。

データの伝送法については、

- (7) The Linear and Interface Circuits Data Book for Design Engineers, Texas Instruments Incorporated, 1973

- (8) Line Interface, TDK-Fairchild 技術情報, 1975

などが役立つでしょう。こういうところは現在のところメーカーの資料に頼る以外に方法はないようです。私が使用している IC の資料は、

- (9) The Integrated Circuits Catalog for Design Engineers,
Texas Instrument Incorporated

- (10) Supplement to the TTL Data Book for Design Engineers,
Texas Instrument Incorporated, 1974

- (11) TTL Data Book, Fairchild Semiconductors, 1972

- (12) Designer's Choice Logic Specification Handbook Vol.1,
Signetics Corporation, 1969

- (13) Digital Integrated Circuits, National Semiconductors Corporation, 1974

- (14) 8080 Microcomputer User's Manual, Intel Corporation,

1975

(15) MCS 85 User's Manual, Intel Corporation, 1976

(16) Intel Data Catalog, 1977

などです。これらの資料はメーカーまたは代理店から入手できると思います。
LSI の中味をある程度理解するには、

(17) 別冊サイエンス 特集エレクトロニクス集積回路, 日本経済新聞社,

1976

がよいと思います。もっと専門的な、たとえば物性論的なことまで理解する
のだったら別の参考書が必要になると思いますが、LSIを使う立場では、そ
こまでは必要ないと思います。

マイクロコンピュータそのものに移って、この分野の現状をある程度技
術的な資料とともに太づかみにするには、

(18) 石田, マイクロコンピュータの活かし方, 産報出版, 1977

があります。

マイクロコンピュータの製作事例としては、

(19) 額田, FAMCOM-80の製作と恋人あて機, つくるマイコン第1集,

CQ 出版社, 1977

(20) 橋爪他, パネル付きマイコン "KUMICOM" とソフトウェア, マイコ
ン, 6 月, 1977

(21) 大川, マイクロコンピュータの作り方, 産報出版, 1975

(22) 八木, TC-2 の製作, つくるコンピュータ, CQ 出版社, 1976

(23) 石木, マイコン製作徹底ガイド, インターフェース, 6 月, 1976

(24) 正田他, マイクロプロセッサ制御の設計, 産報出版, 1976

(25) 大河他, Z80製作記, I/O, 1 月, 1977

などがあります。キットの製作報告として、

(26) 菊川, TK-80の製作と応用, つくるマイコン第1集, CQ 出版社,

1976

があります。

ソフトウェアについて述べればきりがないので、ここでは省略しますが、8080系のソフトウェア構造にはじめて触れる人は、次の書物をあらかじめ読んでおいて下さい。

(27) 大川，マイクロコンピュータ・プログラムの作り方，産報出版，1976
インターフェース回路については文献が多くないのですが，

(28) 村田，マイクロコンピュータの本格的応用，CQ出版社，1977
があります。

応用については，4004の場合ですが，

(29) 多田他，マイクロコンピュータ・システム，共立出版，1976
があります。この他に，

(30) 中川，切り替え弁の制御，インターフェース，6月，1977

(31) 森，ローコスト放電プリンター，I/O，7月，1977

(32) 額田，ラインプリンタ・コントローラの製作，つくるコンピュータ，
CQ出版社，1976

などがありますが，応用事例はまだこれからの分野だと思います。オーディオ用のアンプとして，

(33) 奥沢，ステレオアンプの作り方，日本放送出版協会，1976
があります。

これからもっと，機械要素の制御事例などが発表されなければならないと思います，これはまだ未開拓の分野です。

付 録

8085

● 8085 INSTRUCTION SET

Summary of Processor Instructions

Mnemonic	Description	Instruction Code[1]								Clock[2]	
		D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	Cycles	
MOV r ₁ , r ₂	Move register to register	0	1	D	D	D	S	S	S	4	
MOV M, r	Move register to memory	0	1	1	1	0	S	S	S	7	
MOV r, M	Move memory to register	0	1	D	D	D	1	1	0	7	
HLT	Halt	0	1	1	1	0	1	1	0	5	
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7	
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10	
INR r	Increment register	0	0	D	D	D	1	0	0	4	
DCR r	Decrement register	0	0	D	D	D	1	0	1	4	
INR M	Increment memory	0	0	1	1	0	1	0	0	10	
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10	
ADD r	Add register to A	1	0	0	0	0	S	S	S	4	
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4	
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4	
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4	
ANA r	And register with A	1	0	1	0	0	S	S	S	4	
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4	
ORA r	Or register with A	1	0	1	1	0	S	S	S	4	
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4	
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7	
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7	
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7	
SRB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7	
ANA M	And memory with A	1	0	1	0	0	1	1	0	7	

Mnemonic	Description	Instruction Code[1]								Clock[2]	
		D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	Cycles	
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7	
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7	
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7	
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7	
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7	
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7	
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7	
ANI	And immediate with A	1	1	1	0	0	1	1	0	7	
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7	
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7	
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7	
RLC	Rotate A left	0	0	0	0	0	1	1	1	4	
RRC	Rotate A right	0	0	0	0	1	1	1	1	4	
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4	
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4	
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10	
JC	Jump on carry	1	1	0	1	1	0	1	0	7/10	
JNC	Jump on no carry	1	1	0	1	0	0	1	0	7/10	
JZ	Jump on zero	1	1	0	0	1	0	1	0	7/10	
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	7/10	
JP	Jump on positive	1	1	1	1	0	0	1	0	7/10	
JM	Jump on minus	1	1	1	1	1	0	1	0	7/10	
JPE	Jump on parity even	1	1	1	0	1	0	1	0	7/10	
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	7/10	
CALL	Call unconditional	1	1	0	0	1	1	0	1	18	
CC	Call on carry	1	1	0	1	1	1	0	0	9/18	
CNC	Call on no carry	1	1	0	1	0	1	0	0	9/18	
CZ	Call on zero	1	1	0	0	1	1	0	0	9/18	
CNZ	Call on no zero	1	1	0	0	0	1	0	0	9/18	
CP	Call on positive	1	1	1	1	0	1	0	0	9/18	
CM	Call on minus	1	1	1	1	1	1	0	0	9/18	

Mnemonic	Description	Instruction Code[1]								Clock[2]
		D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	Cycles
CPE	Call on parity even	1	1	1	0	1	1	0	0	9/18
CPO	Call on parity odd	1	1	1	0	0	1	0	0	9/18
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	6/12
RNC	Return on no carry	1	1	0	1	0	0	0	0	6/12
RZ	Return on zero	1	1	0	0	1	0	0	0	6/12
RNZ	Return on no zero	1	1	0	0	0	0	0	0	6/12
RP	Return on positive	1	1	1	1	0	0	0	0	6/12
RM	Return on minus	1	1	1	1	1	0	0	0	6/12
RPE	Return on parity even	1	1	1	0	1	0	0	0	6/12
RPO	Return on parity odd	1	1	1	0	0	0	0	0	6/12
RST	Restart	1	1	A	A	A	1	1	1	12
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	12
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	12
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	12
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	12
POP B	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10

Mnemonic	Description	Instruction Code[1]								Clock[2]	
		D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	Cycles	
STA	Store A direct	0	0	1	1	0	0	1	0	13	
LDA	Load A direct	0	0	1	1	1	0	1	0	13	
XCHG	Exchange D & E, H & L	1	1	1	0	1	0	1	1	4	
	Registers										
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	16	
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	6	
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	6	
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10	
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10	
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10	
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10	
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7	
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7	
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7	
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7	
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	6	
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	6	
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	6	
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	6	
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	6	
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	6	
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	6	
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	6	
CMA	Complement A	0	0	1	0	1	1	1	1	4	
STC	Set carry	0	0	1	1	0	1	1	1	4	
CMC	Complement carry	0	0	1	1	1	1	1	1	4	
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4	
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16	
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16	
EI	Enable Interrupts	1	1	1	1	1	0	1	1	4	
DI	Disable interrupt	1	1	1	1	0	0	1	1	4	
NOP	No-operation	0	0	0	0	0	0	0	0	4	
RIM	Read Interrupt Mask	0	0	1	0	0	0	0	0	4	
SIM	Set Interrupt Mask	0	0	1	1	0	0	0	0	4	

- NOTES: 1. DDD or SSS-000 B-001 C-010 D-011 E-100 H-101 L-110 Memory-111A.
 2. Two possible cycle times, (6/12) indicate instruction cycles dependent on condition flags.

●ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias..... 0°C to 70°C
 Storage Temperature..... -65°C to $+150^{\circ}\text{C}$
 Voltage on Any Pin
 With Respect to Ground..... -0.3 to $+7\text{V}$
 Power Dissipation..... 1.5Watt

*COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

●D. C. CHARACTERISTICS

($T_A=0^{\circ}\text{C}$ to 70°C ; $V_{cc}=5\text{V} \pm 5\%$; $V_{ss}=0\text{V}$; unless otherwise specified)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	$+0.8$	V	
V_{IH}	Input High Voltage	2.0	$V_{cc}+0.5$	V	
V_{OL}	Output Low Voltage		0.45	V	$I_{OL}=2\text{mA}$
V_{OH}	Output High Voltage	2.4		V	$I_{OH}=-400\mu\text{A}$
I_{CC}	Power Supply Current		170	mA	
I_{IL}	Input Leakage		± 10	μA	$V_{in}=V_{cc}$
I_{IH}	Output Leakage		± 10	μA	$0.45\text{V} \leq V_{out} \leq V_{cc}$
V_{ILR}	Input Low Level, RESET	-0.5	$+0.8$	V	
V_{IHR}	Input High Level, RESET	2.4	V_{cc}	V	
V_{HY}	Hysteresis, RESET	0.25		V	

Bus Timing Specification as a T_{cyc}
Dependent

t_{AL}	—	$(1/2)T - 50$	MIN
t_{LA}	—	$(1/2)T - 20$	MIN
t_{LL}	—	$(1/2)T - 40$	MIN
t_{LCK}	—	$(1/2)T - 50$	MIN
t_{LC}	—	$(1/2)T - 30$	MIN
t_{AD}	—	$(5/2 + N)T - 225$	MAX
t_{RD}	—	$(3/2 + N)T - 200$	MAX
t_{RAE}	—	$(1/2)T - 60$	MIN
t_{CA}	—	$(1/2)T - 40$	MIN
t_{DW}	—	$(3/2 + N)T - 60$	MIN
t_{WD}	—	$(1/2)T - 80$	MIN
t_{CC}	—	$(3/2 + N)T - 80$	MIN
t_{CL}	—	$(1/2)T - 110$	MIN
t_{ARY}	—	$(3/2)T - 260$	MAX
t_{HACK}	—	$(1/2)T - 50$	MIN
t_{HABF}	—	$(1/2)T + 30$	MAX
t_{HABE}	—	$(1/2)T + 30$	MAX
t_{AC}	—	$(2/2)T - 50$	MIN
t_1	—	$(1/2)T - 80$	MIN
t_2	—	$(1/2)T - 40$	MIN
t_{RV}	—	$(3/2)T - 80$	MIN
t_{INS}	—	$(1/2)T + 200$	MIN

NOTE: N is equal to the total WAIT states.

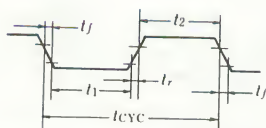
$$T = t_{cyc}.$$

●A. C. CHARACTERISTICS

(T_A = 0°C to 70°C; V_{CC} = 5 V ± 5 %; V_{SS} = 0 V)

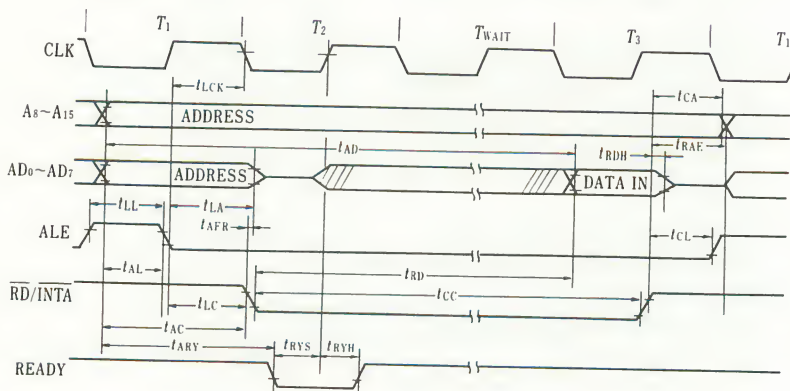
Symbol	Parameter	Min.	Max.	Units	Test Conditions
T _{cyC}	CLK Cycle Period	320	2000	ns	See notes 1, 2, 3, 4, 5 T _{cyC} = 320ns; C _L = 150pF
t _l	CLK Low Time	80		ns	
t _h	CLK High Time	120		ns	
t _r , t _f	CLK Rise and Fall Time		30	ns	
t _{VL}	Address Valid Before Trailing Edge of ALE	110		ns	
t _{LA}	Address Hold Time After ALE	100		ns	
t _{L.L}	ALE Width	120		ns	
t _{LCK}	ALE Low During CLK High	100		ns	
t _{LC}	Trailing Edge of ALE to Leading Edge of Control	130		ns	
t _{AFR}	Address Float After Leading Edge of READ (INTA)			ns	
t _{AD}	Valid Address to Valid Data In		0	ns	
t _{RD}	READ (or INTA) to Valid Data		575	ns	
t _{RDH}	Data Hold Time After READ (INTA)	0	280	ns	
t _{RAE}	Trailing Edge of READ to Re-Enabling of Address	100		ns	
t _{CA}	Address (A8-A15) Valid After Control	120		ns	
t _{DW}	Data Valid to Trailing Edge of WRITE	420		ns	
t _{WD}	Data Valid After Trailing Edge of WRITE	80		ns	
t _{CC}	Width of Control Low (RD, WR, INTA)	400		ns	
t _{CL}	Trailing Edge of Control to Leading Edge of ALE	50		ns	
t _{ARY}	READY Valid From Address Valid		220	ns	
t _{RYS}	READY Setup Time to Leading Edge of CLK	110		ns	
t _{RYH}	READY Hold Time	0		ns	
t _{HACK}	HLDA Valid to Trailing Edge of CLK	110		ns	
t _{HAF}	Bus Float After HLDA		190	ns	
t _{RV}	Control Trailing Edge to Leading Edge of Next Control	400		ns	
t _{AC}	Address Valid to Leading Edge of Control	270		ns	
t _{HDS}	HOLD Setup Time to Trailing Edge of CLK	170		ns	
t _{HDD}	HOLD Hold Time	0		ns	
t _{INS}	INTR Setup Time to Leading Edge of CLK (M1, T1 only). Also RST and TRAP	360		ns	
t _{INH}	INTR Hold Time	0		ns	

- NOTES: 1. A8-15 Address Specs apply to $\text{IO}/\overline{\text{M}}$, S0 and S1. □
2. For all output timing where $C_L \neq 150\text{pF}$ use the following correction factors:
 $25\text{pF} \leq C_L < 150\text{pF}$: $- .10 \text{ ns/pF}$
 $150\text{pF} < C_L \leq 300\text{pF}$: $+ .30 \text{ ns/pF}$
3. Output timings are measured with purely capacitive load.
4. All timings are measured at output voltage $V_I = 8 \text{ V}$, $V_H = 2.0 \text{ V}$, and 1.5 V with 20ns rise and fall time on inputs.
5. To calculate timing specifications at other values of T_{CYC} use the table in Table (p.216).
6. L.E.=Leading Edge T.E.=Trailing Edge



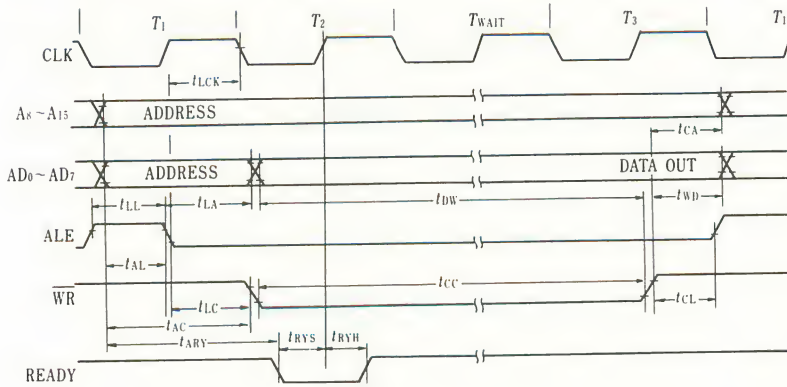
付図 1 CLOCK TIMING
WAVEFORM

READ OPERATION



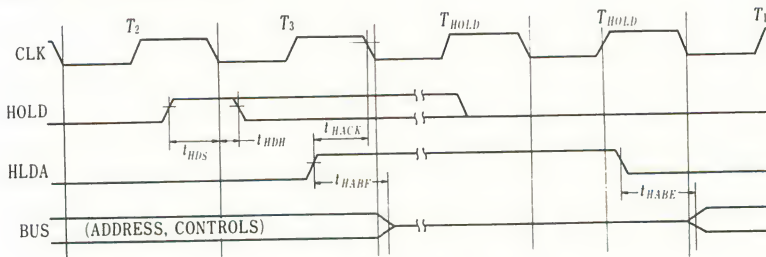
付図 2 8085 BUS TIMING(1)

WRITE OPERATION



付図 2 8085 BUS TIMING(2)

HOLD OPERATION



付図 2 8085 BUS TIMING(3)

●SUMMARY OF 8085 INSTRUCTIONS

INSTR	STATES	CYCLES	NOTES	INSTR	STATES	CYCLES	NOTES
AR r	4	1		OUT	10	3	
AR I	7	2		PCHL	6	1	2
AR M	7	2		POP	10	3	
CALL	18	5	2	PUSH	12	3	2
CCN	9/18	2/5	2,4	ROT	4	1	
CMC	4	1		RET	10	3	
CMA	4	1		RCN	6/12	1/3	2
DAA	4	1		RIM*	4	1	
DAD	10	3		RST	12	3	2
DCR r	4	1	3	SHLD	16	5	
DCR M	10	3		SIM*	4	1	
DCX	6	1	2	STA	13	4	
DI	4	1		STAX	7	2	
EI	4	1		STC	4	1	
HLT	5	1	5	XCHG	4	1	
IN	10	3		XTHL	16	5	
INR r	4	1	3	SPHL	6	1	2
INR M	10	3					
INX	6	1	2				
JMP	10	3					
JCN	7/10	2/3	4				
LDS	13	4					
LDAX	7	2					
LHLD	16	5					
LXU	10	3					
MVI M	10	3					
MVI r	7	2					
MOV M, r	7	2					
MOV r, M	7	2					
MOV r, r	4	1	3				
NOP	4	1					

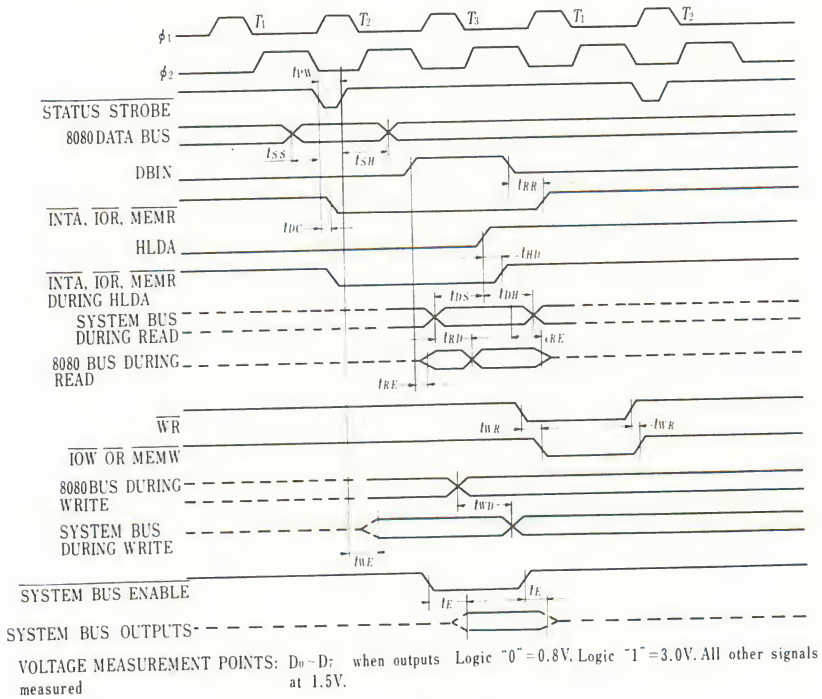
*New Instruction

NOTES:

1. Two possible state-cycle times indicates dependence on condition flag.
2. Increase in states over 8080 due to necessity of a T_6 during M_1 .
3. Decreased from 5 to 4 states during M_1 .
4. Instruction branches over last address fetch if condition false.
5. 5 cycles to get a HLT state. 1 cycle necessary to get out a HLT.

SCHOTTKY BIPOLAR 8228

● WAVEFORMS

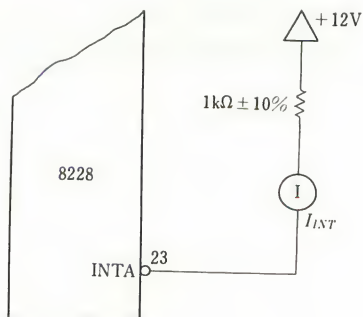


付図 3

●A. C. Characteristics

 $(T_A = 0^\circ\text{C to } 70^\circ\text{C}; V_{CC} = 5\text{V} \pm 5\%)$

Symbol	Parameter	Limits		Units	Condition
		Min.	Max.		
t_{PW}	Width of Status Strobe	22		ns	
t_{SS}	Setup Time, Status Inputs D_0 - D_7	8		ns	
t_{SH}	Hold Time, Status Inputs D_0 - D_7	5		ns	
t_{DC}	Delay from $\overline{\text{STSTB}}$ to any Control Signal	20	60	ns	$C_L = 100\text{pF}$
t_{RR}	Delay from $\overline{\text{DBIN}}$ to Control Outputs		30	ns	$C_L = 100\text{pF}$
t_{RE}	Delay from $\overline{\text{DBIN}}$ to Enable/Disable 8080 Bus		45	ns	$C_L = 25\text{pF}$
t_{RD}	Delay from System Bus to 8080 Bus during Read		30	ns	$C_L = 25\text{pF}$
t_{WR}	Delay from $\overline{\text{WR}}$ to Control Outputs	5	45	ns	$C_L = 100\text{pF}$
t_{WE}	Delay to Enable System Bus $\overline{\text{DB}_0}$ - $\overline{\text{DB}_7}$ after $\overline{\text{STSTB}}$		30	ns	$C_L = 100\text{pF}$
t_{WD}	Delay from 8080 Bus D_0 - D_7 to System Bus $\overline{\text{DB}_0}$ - $\overline{\text{DB}_7}$ during Write	5	40	ns	$C_L = 100\text{pF}$
t_E	Delay from System Bus Enable to System Bus $\overline{\text{DB}_0}$ - $\overline{\text{DB}_7}$		30	ns	$C_L = 100\text{pF}$
t_{HD}	HLDA to Read Status Outputs		25	ns	
t_{DS}	Setup Time, System Bus Inputs to HLDA	10		ns	
t_{DH}	Hold Time, System Bus Inputs to HLDA	20		ns	$C_L = 100\text{pF}$



付図 4

●D. C. Characteristics

(T_A=0 °C to 70 °C; V_{cc}= 5 V ± 5 %)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.[1]	Max.		
V _c	Input Clamp Voltage, All Input		0.75	-1.0	V	V _{cc} =4.75V; I _c = - 5 mA
I _i	Input Load Current, STSTB			500	μA	V _{cc} =5.25V V _r =0.45V
	D ₂ & D ₆			750	μA	
	D ₀ , D ₁ , D ₄ , D ₅ & D ₇			250	μA	
	All Other Inputs			250	μA	
I _k	Input Leakage Current STSTB			100	μA	V _{cc} =5.25V V _r =5.25V
	DB ₀ -DB ₇			20	V	
	All Other Inputs			100	mA	
V _{TH}	Input Threshold Voltage, All Inputs	0.8		2.0	V	V _{cc} = 5 V
I _{cc}	Power Supply Current		140	190	mA	V _{cc} =5.25V
V _{OL}	Output Low Voltage, D ₀ -D ₇			0.45	V	V _{cc} =4.75V; I _{OL} = 2 mA
	All Other Outputs			0.45	V	I _{OL} =10mA
V _{OH}	Output High Voltage, D ₀ -D ₇	3.6	3.8		V	V _{cc} =4.75V; I _{OH} = -10μA
	All Other Outputs	2.4			V	I _{OH} = - 1 mA
I _{OS}	Short Circuit Current, All Outputs	15		90	mA	V _{cc} = 5 V
I _{OFF}	Off State Output Current, All Control Outputs			100	μA	V _{cc} =5.25V; V _O = 5.25V
				-100	μA	V _O =0.45V
I _{INT}	INTA Current			5	mA	(See Figure below)

Note 1 : Typical values are for T_A=25 °C and nominal supply voltages.

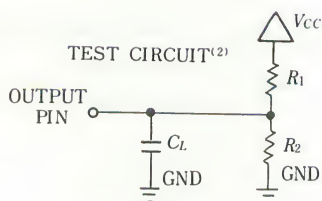
●Capacitance

This parameter is periodically sampled and not 100% tested.

Symbol	Parameter	Limits			Unit
		Min.	Typ. [1]	Max.	
C_{IN}	Input Capacitance		8	12	pF
C_{OUT}	Output Capacitance Control Signals		7	15	pF
I/O	I/O Capacitance (D or DB)		8	15	pF

TEST CONDITIONS: $V_{BIAS}=2.5V$, $V_{CC}=5.0V$, $T_A=25^{\circ}C$,
 $f=1\text{ MHz}$.

Note 2: For D₆-D₇: $R_1=4\text{ k}\Omega$, $R_2=\infty\Omega$,
 $C_L=25\text{ pF}$. For all other outputs:
 $R_1=500\Omega$, $R_2=1\text{ k}\Omega$, $C_L=100\text{ pF}$.



付図 5

索引

《あ 行》

I/O読出し	39
INTA	53
RST n	54
RST 6.5	63
RST 7.5	63
RST 命令	114
アクティブプルアップ	19
アドレスバス	14
アルゴリズム	147
EI 命令	51
印刷サブルーチン	84
印字動作の終了	81
印字ヘッド	127
印字ヘッドの駆動部	127
ACKNLG信号	131
M ₁ サイクル	52
SN74367	70

《か 行》

書込みのタイミング	12
紙テープパンチ動作の完了	88
紙テープパンチのインターフェース回路	86
紙テープ読込み	92
キーボード	92
キーボード読込み	94
キーボードロック	95
キャリッジリターン	83
逆能動領域	18
禁止入力	23

9 3 1 8	115
クロック	36
クロック線	34
クロックの周期	36
光源	138
光電式ロータリーエンコーダ	175
高速プリンタ	126
高速プリンタのインターフェース回路	133
誤動作	200

《さ 行》

3 状態素子	23
サイクルスレーシング	34
最下位のビット	11
最上位のビット	11
雑音余裕	19
シンク電流	19
時間待ち	169
時分割	27
手動式紙テープリーダ	136
手動式光電紙テープリーダの インターフェース回路	146
周波数特性	159
出力ポート	43
情報の単位	11
スタック	54
ステータス	84
ストローブパルス	130
スプロケット信号のチャタリング	145
スプロケット信号の発振	145
セットアップ時間	12

正極性のパルス	13
整形増幅	177
ソース電流	18
双方向性のデータバス	30
増幅回路	139

《た 行》

タイプコマンド	78
タイプライタ	75
タイムシェア	27
タイムマルチプレックス	27
ダーリントトランジスタ	25
チャタリング	81
直流モータ	157
ツイストペア	130
DA変換回路	157, 160
DM8837	98
DMA (direct memory access)	33, 185
TTLゲートの動作原理	17
データのマルチプレックス	25
データバス	10
低周波発振器	160
低速バス	97
電圧変換回路	82
電気自動車	153
電気自動車の速度	173
電力増幅用サーボアンプ	157
TRAP	62, 156
TRAPプログラム	172
トータムボール	19
動作の完了信号	131

《な 行》

ノイズ	200
-----	-----

《は 行》

8 単位紙テープの寸法	140
8085のバス	59
バス切り離し完了信号	33
バス制御線	31
パルス	45
パンチコマンド	86
パンチサブルーチン	89
非重複2相クロック	36
非常停止プログラム	172
標準TTL素子の等価回路	17
フォトダイオード	138
フォトダイオードの増幅回路	177
フロントパネル	156
符号化	14
HOLD要求	190
ホールド時間	12
負極性の信号	13

《ま 行》

マザーボード	67
マシンインターフェースの問題	95
マシンサイクル	37
マスク不可能な割込み	62
マルチエミッタ	17
マンマシンインターフェースの問題	151
memory mapped I/O	48
メモリへの書込み	39
メモリ読出し	39

《や 行》

ユニバーサル基板	67
優先度	118

《ら 行》

ラインプリンタ	126
リードコマンド	90

リアルタイムクロック.....	121
リセット線.....	34
リレー.....	75
リレードライバ回路.....	76

割込み処理.....	110
割込み処理ルーチン.....	134
割込み要求.....	111
割込み要求線.....	51

《わ 行》

割込み.....	32, 51
----------	--------

マイクロインターフェースの作り方 《電子科学シリーズ》 ⑦⑤
コンピュータ

1977 年 11 月 15 日 初版印刷

1977 年 11 月 25 日 初版発行

著 者 大 川 善 邦

発 行 者 松 山 邦 雄

発 行 所 産報出版株式会社
東京都港区浜松町 1-10-17 (郵便番号105)
電話東京(03)436-4151大代表・振替東京1-58204

〈検印廃止〉

印 刷=廣濟堂印刷株式会社

製 本=秋元製本株式会社

© Printed in Japan, 1977

万一、乱丁・落丁がございましたら書店または発行所でお取換えいたします。

定価はカバーに明記してあります

2355-130275(01)-2779

トップをやさしくおくる……

好評 《電子科学シリーズ》 出版案内

— A 5 判・送料各巻 180 円 —

- ① トランジスタの使い方……………垂井康夫・手島寛郎共著●1,300円
- ② やさしい電子計算機……………渡部弘之著●2,200円
- ③ 安定化電源回路の実際……………大塚 敏著●1,500円
- ④ トランジスタパルス回路……………小柴典居著●1,880円
- ⑤ トランジスタ増幅回路の設計法……………片方善治著●1,600円
- ⑥ やさしい光電素子の応用……………山中 一郎著●1,000円
- ⑦ やさしい超音波の応用……………藤森聡雄著●1,800円
- ⑧ わかりやすい半導体光物性……………高木克己・山田祥二共著●2,500円
- ⑨ エサキ・ダイオード〈その特性と使い方〉……………天野橋太郎著●1,400円
- ⑩ わかりやすいプログラミング1〈ALGOL COBOLの理解と演習〉……………岡崎嘉春・角山榮治共著●1,500円
- ⑪ トランジスタ高周波回路の基礎……………風間茂穂著● 900円
- ⑫ 実用トランジスタ増幅回路集……………伊藤祐弥編●1,200円
- ⑬ トランジスタビデオ回路の設計法……………片方善治著●1,200円
- ⑭ わかりやすいシンクロスコープ測定法……………塩沢政美著●1,600円
- ⑮ トランジスタOTL回路の基礎……………鴨治儀秋著●1,100円
- ⑯ エレクトロニクス製図法……………田中兼義・片岡徳昌共著●1,300円
- ⑰ トランジスタOTL回路の設計法……………鴨治儀秋著●1,000円
- ⑱ トランジスタディジタル回路……………細田祝資著●1,400円
- ⑲ トランジスタ直流増幅器……………戸室晃一著●1,800円
- ⑳ わかりやすいブリッジ回路……………原 宏著●1,300円
- ㉑ マイクロ波回路の基礎知識……………小西良弘著●1,400円
- ㉒ トランジスタDA・AD変換器……………今井 聖著●1,600円
- ㉓ わかりやすい集積回路……………関口存哉・田内省二共著●1,200円
- ㉔ わかりやすいエレクトロニクス計測法……………今井 嵩著●1,600円
- ㉕ トランジスタ高周波回路の実際……………風間茂穂著●1,300円
- ㉖ わかりやすいFM技術……………伊藤祐弥著●1,100円
- ㉗ MOS-ICとFET……………山崎英蔵・太久保利美共著●2,200円
- ㉘ ビデオレコーディング技術……………木原信敏著●1,980円
- ㉙ わかりやすいプログラミング2〈COBOL文法の実際〉……………ADSコボル研究課編●1,300円
- ㉚ わかりやすいプログラミング3〈フォートランの文法と実際〉……………田中武二・酒井敏・松尾隆雄共著●1,200円
- ㉛ わかりやすいプログラミング4〈PL/I入門〉……………竹下 亨著●1,500円
- ㉜ レーザーとその応用……………島津備愛著●1,800円
- ㉝ サイリスタの応用……………小津厚二郎・横田博共著●2,200円
- ㉞ 半導体ICの使い方……………小林貞男・初鹿野凱一共著●1,200円
- ㉟ 基本コンピュータの設計……………猪飼国夫著●1,000円
- ㊱ COMとその周辺機器……………近藤敏一郎編●1,500円

■定価は重版発行の際、変更することもありますのであらかじめ御諒下下さい。

- 37 IC機器の設計……………千葉幸正著●1,800円
- 38 わかりやすいME<医用電子技術>……………瓜谷富三著●2,200円
- 39 わかりやすい周辺装置……………井上武彦著●1,900円
- 40 わかりやすいプログラミング5<RPG入門>……………安井康晴著●1,200円
- 41 電子計算機の基礎理論<論理設計と解析>……………山本英雄・市川忠男共著●1,300円
- 42 固体発光素子とその応用……………中村哲郎・内丸清共著●1,600円
- 43 ICによる測定器の製作……………久賀八洲男著●1,500円
- 44 パルス計測の基礎と応用……………山中英夫著●1,700円
- 45 サーボ機器の実際<オーディオ・ビデオへの応用>……………本原信敏編●1,400円
- 46 ファクシミリとその応用……………窪田啓次郎著●1,800円
- 47 わかりやすいパワー技術……………杉田 稔著●1,200円
- 48 コンピュータ方式の設計……………発田 弘著●2,200円
- 49 サイリスタの応用装置……………河野薫勝著●1,300円
- 50 定電圧ICとその使い方……………伝田精一・立川巖共著●1,800円
- 51 感温半導体の実際……………二本久夫著●1,900円
- 52 アクティブフィルタの設計……………柳沢健・金光磐共著●1,500円
- 53 カウンタ回路とその応用……………志賀正明著●1,200円
- 54 マイクロプログラミングとその応用……………上原 一・松崎稔共著●1,860円
- 55 フィルタの理論と設計……………柳沢健・神林紀嘉共著●2,200円
- 56 放送用SHF受信機の設計……………小西良弘著●1,900円
- 57 プリント配線の接続設計……………網島瑛一著●1,750円
- 58 わかりやすい通信網……………横井 満著●1,560円
- 59 リニア回路の設計……………藤井信生著●1,380円
- 60 MOS-LSIとその応用……………山崎英蔵編●1,860円
- 61 マイクロコンピュータの使い方……………石田晴久著●1,300円
- 62 ICメモリの使い方……………新田松雄・大表良一共著●1,400円
- 63 高速パルス技術の実際……………大平隆夫著●1,300円
- 64 マイクロコンピュータの作り方……………大川善邦編著●1,600円
- 65 マイクロコンピュータの開発技法……………石田 芳著●1,580円
- 66 マイクロプロセッサ制御の設計……………正田英介・田村 稔共著●1,300円
- 67 マイクロコンピュータ機器の設計……………千葉幸正著●1,500円
- 68 論理回路の故障診断……………安居院猛・内藤祥雄共著●1,500円
- 69 PCM通信の技術……………金子高志著●1,600円
- 70 PLL-ICの使い方……………畑 雅恭・古川計介共著●1,900円
- 71 CMOSの応用技法……………鈴木八十二著●1,900円
- 72 マイクロコンピュータ・プログラムの作り方……………大川善邦著●1,500円
- 73 マイクロコンピュータの活かし方……………石田晴久著●1,600円
- 74 演算増幅器回路の設計……………藤井信生著●1,900円
- 75 マイクロコンピュータ・インターフェースの作り方……………大川善邦著●1,800円

《続刊予定》

発振回路と応用技術……………細田祝資著● 円

産報出版・案内

■定価は重版発行の際、変更することもありますのであらかじめ御諒承下さい。

好評・電子科学シリーズの姉妹編…

電子科学《演習シリーズ》出版案内

《演習シリーズ》は好評を博している《電子科学シリーズ》の姉妹編として、電子技術各分野の実務と実習を主眼に企画・執筆・編集されたものです。

職場・教室・研究室のテキストにまた自習書に、簡潔な解説は既得知識の整理に役立ち、豊かな例題とその運用処理はダイナミックなトレーニング・グラウンドをあなたに提供します。

〈各巻共A 5判 円180円〉

- ① **トランジスタパルス回路設計演習**……………小柴典居編●1,300円
設計現場で日常ぶつかる問題を集め、具体的な解説を行なった。
- ② **ALGOLの演習**……………岡崎・角山・林共著●1,200円
「初歩から実務まで」を主眼に、詳しくアルゴルについて説明。
- ③ **電子回路設計のための数学演習**……………塩沢・鍛冶共著●1,300円
電子工学と数学の関係を有機的に把握できるよう総合的に解説。
- ④ **COBOLの演習**……………浜田・佐柳・松下共著●960円
実務においてよく使用されるパターンのものを選んだ実践的演習。
- ⑤ **FORTRANの演習**……………松尾・若松・山下共著●1,800円
多くの具体的な問題によってフォートランを十分に使いこなせる。
- ⑥ **デジタル回路設計演習**……………細田悦資著●1,600円
パルス回路からブール代数、デジタル回路の論理設計とその応用。
- ⑦ **フローチャートの演習**……………石丸勝敏著●1,600円
フローチャートを書くに当っての手順、論理展開を平易に説く。
- ⑧ **ミニコン・プログラムの演習**……………秋山稔編●1,200円
ミニコン専門のプログラム学習書として入門者向けにやさしく解説。
- ⑨ **IC機器の設計演習**……………千葉幸正著●1,200円
仕様から製作までを一つの流れとして演習を行なう画期的書。
- ⑩ **デジタルICの基本演習**……………久賀八州男著●1,200円
デジタルICの応用をわかりやすく实际的に記述したユニークな書。
- ⑪ **四則演算回路の設計演習**……………加藤隆明・松尾博共著●1,400円
演算回路の論理的アプローチと実際回路との総合的把握に役立つ。
- ⑫ **シミュレーションの演習**……………岸田孝一他著●1,500円
方法論のスペシャリストを目指すプログラマのための入門書。
- ⑬ **パルス・デジタル回路の設計演習**……………齊藤尚武著●1,980円
基本回路の動作原理を理解すると同時に回路設計ができるよう編集。
- ⑭ **パルス回路設計演習**……………小柴典居・清水賢資共著●1,850円
パルス回路の数学的取扱い法と基礎回路の設計の双方を解説。

■定価は重版発行の際、変更することもありますのであらかじめ御諒承下さい。

産報出版・案内



マイクロー
インターフェースの作り方

大川 善邦 著



大川善邦（おおかわ よしくに）

1934年7月生れ／1959年3月東京
大学工学部機械工学科卒業／1964
年3月同大学院博士課程卒業

現職＝岐阜大学工学部教授

専門＝計測および制御

著書＝マイクロコンピュータの作
り方、マイクロコンピュータプロ
グラムの作り方(産報出版)、イン
フォメーション(日本経済新聞)、ミ
クロプログラミング(培風館)ほか

マイクロコンピュータが各界で絶賛されている。しかし、マイクロコンピュータの“素晴らしさ”は、最高度に集積されたLSIのチップの中にあるのではなく、マイクロコンピュータで何が出来てくるかを具体的に考え出す人間の頭脳の中から生まれてくるのである。

電子科学シリーズ

巻
号
目
次

トランジスタの使い方	1	121. プログラムの作り方	72
トランジスタ・バルス回路	4	122. インターフェースの作り方	75
高速・バルス技術の実例	63	123. 実用モニタとアセンブラ	78
124. デジタル回路	18	124. CRディスプレイ技法	79
125. DA・AD変換器	22	125. PLAの使い方	80
リニア回路の設計	59	126. ICメモリの使い方	82
演算増幅器回路の設計	74	127. CMOSの応用技法	71
128. 増幅回路の設計法	8	128. MOS-LSIとその応用	60
実用129. 増幅回路集	12	129. MOS-ICとFET	27
130. TTL回路の基礎	15	130. 半導体ICの使い方	34
131. TTL回路の設計法	17	131. 集積回路	29
132. 高速増幅器	19	132. IC機器の設計	37
133. 意図回路の基礎	11	133. カウンタ回路とその応用	53
134. 意図回路の実例	28	134. 論理回路の故障診断	68
135. ビデオ回路の設計法	13	135. エサキ・ダイオード	9
ビデオレコーディング技術	28	136. 定電圧ICとその使い方	50
137. フリクション回路	20	137. 安全止電源回路の実例	3
マイクロ回路の基礎知識	21	138. やさしい電子計算機	2
フィルタの理論と設計	55	139. 基本コンピュータの設計	35
アクティブフィルタの設計	52	140. コンピュータ方式の設計	48
PLL-ICの使い方	70	141. 電子計算機の基礎理論	41
電源回路と変換技術	76	142. わかりやすい周辺装置	39
放送用SHF受信機の設計	56	143. COMとその周辺機器	36
わかりやすい通信機	58	144. マイクロプログラミングとその応用	54
PCM通信の技術	69	145. マイクロプログラミング (1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26) (27) (28) (29) (30) (31) (32) (33) (34) (35) (36) (37) (38) (39) (40) (41) (42) (43) (44) (45) (46) (47) (48) (49) (50) (51) (52) (53) (54) (55) (56) (57) (58) (59) (60) (61) (62) (63) (64) (65) (66) (67) (68) (69) (70) (71) (72) (73) (74) (75) (76) (77) (78) (79) (80) (81) (82) (83) (84) (85) (86) (87) (88) (89) (90) (91) (92) (93) (94) (95) (96) (97) (98) (99) (100) (101) (102) (103) (104) (105) (106) (107) (108) (109) (110) (111) (112) (113) (114) (115) (116) (117) (118) (119) (120) (121) (122) (123) (124) (125) (126) (127) (128) (129) (130) (131) (132) (133) (134) (135) (136) (137) (138) (139) (140) (141) (142) (143) (144) (145) (146) (147) (148) (149) (150) (151) (152) (153) (154) (155) (156) (157) (158) (159) (160) (161) (162) (163) (164) (165) (166) (167) (168) (169) (170) (171) (172) (173) (174) (175) (176) (177) (178) (179) (180) (181) (182) (183) (184) (185) (186) (187) (188) (189) (190) (191) (192) (193) (194) (195) (196) (197) (198) (199) (200) (201) (202) (203) (204) (205) (206) (207) (208) (209) (210) (211) (212) (213) (214) (215) (216) (217) (218) (219) (220) (221) (222) (223) (224) (225) (226) (227) (228) (229) (230) (231) (232) (233) (234) (235) (236) (237) (238) (239) (240) (241) (242) (243) (244) (245) (246) (247) (248) (249) (250) (251) (252) (253) (254) (255) (256) (257) (258) (259) (260) (261) (262) (263) (264) (265) (266) (267) (268) (269) (270) (271) (272) (273) (274) (275) (276) (277) (278) (279) (280) (281) (282) (283) (284) (285) (286) (287) (288) (289) (290) (291) (292) (293) (294) (295) (296) (297) (298) (299) (300) (301) (302) (303) (304) (305) (306) (307) (308) (309) (310) (311) (312) (313) (314) (315) (316) (317) (318) (319) (320) (321) (322) (323) (324) (325) (326) (327) (328) (329) (330) (331) (332) (333) (334) (335) (336) (337) (338) (339) (340) (341) (342) (343) (344) (345) (346) (347) (348) (349) (350) (351) (352) (353) (354) (355) (356) (357) (358) (359) (360) (361) (362) (363) (364) (365) (366) (367) (368) (369) (370) (371) (372) (373) (374) (375) (376) (377) (378) (379) (380) (381) (382) (383) (384) (385) (386) (387) (388) (389) (390) (391) (392) (393) (394) (395) (396) (397) (398) (399) (400) (401) (402) (403) (404) (405) (406) (407) (408) (409) (410) (411) (412) (413) (414) (415) (416) (417) (418) (419) (420) (421) (422) (423) (424) (425) (426) (427) (428) (429) (430) (431) (432) (433) (434) (435) (436) (437) (438) (439) (440) (441) (442) (443) (444) (445) (446) (447) (448) (449) (450) (451) (452) (453) (454) (455) (456) (457) (458) (459) (460) (461) (462) (463) (464) (465) (466) (467) (468) (469) (470) (471) (472) (473) (474) (475) (476) (477) (478) (479) (480) (481) (482) (483) (484) (485) (486) (487) (488) (489) (490) (491) (492) (493) (494) (495) (496) (497) (498) (499) (500) (501) (502) (503) (504) (505) (506) (507) (508) (509) (510) (511) (512) (513) (514) (515) (516) (517) (518) (519) (520) (521) (522) (523) (524) (525) (526) (527) (528) (529) (530) (531) (532) (533) (534) (535) (536) (537) (538) (539) (540) (541) (542) (543) (544) (545) (546) (547) (548) (549) (550) (551) (552) (553) (554) (555) (556) (557) (558) (559) (560) (561) (562) (563) (564) (565) (566) (567) (568) (569) (570) (571) (572) (573) (574) (575) (576) (577) (578) (579) (580) (581) (582) (583) (584) (585) (586) (587) (588) (589) (590) (591) (592) (593) (594) (595) (596) (597) (598) (599) (600) (601) (602) (603) (604) (605) (606) (607) (608) (609) (610) (611) (612) (613) (614) (615) (616) (617) (618) (619) (620) (621) (622) (623) (624) (625) (626) (627) (628) (629) (630) (631) (632) (633) (634) (635) (636) (637) (638) (639) (640) (641) (642) (643) (644) (645) (646) (647) (648) (649) (650) (651) (652) (653) (654) (655) (656) (657) (658) (659) (660) (661) (662) (663) (664) (665) (666) (667) (668) (669) (670) (671) (672) (673) (674) (675) (676) (677) (678) (679) (680) (681) (682) (683) (684) (685) (686) (687) (688) (689) (690) (691) (692) (693) (694) (695) (696) (697) (698) (699) (700) (701) (702) (703) (704) (705) (706) (707) (708) (709) (710) (711) (712) (713) (714) (715) (716) (717) (718) (719) (720) (721) (722) (723) (724) (725) (726) (727) (728) (729) (730) (731) (732) (733) (734) (735) (736) (737) (738) (739) (740) (741) (742) (743) (744) (745) (746) (747) (748) (749) (750) (751) (752) (753) (754) (755) (756) (757) (758) (759) (760) (761) (762) (763) (764) (765) (766) (767) (768) (769) (770) (771) (772) (773) (774) (775) (776) (777) (778) (779) (780) (781) (782) (783) (784) (785) (786) (787) (788) (789) (790) (791) (792) (793) (794) (795) (796) (797) (798) (799) (800) (801) (802) (803) (804) (805) (806) (807) (808) (809) (810) (811) (812) (813) (814) (815) (816) (817) (818) (819) (820) (821) (822) (823) (824) (825) (826) (827) (828) (829) (830) (831) (832) (833) (834) (835) (836) (837) (838) (839) (840) (841) (842) (843) (844) (845) (846) (847) (848) (849) (850) (851) (852) (853) (854) (855) (856) (857) (858) (859) (860) (861) (862) (863) (864) (865) (866) (867) (868) (869) (870) (871) (872) (873) (874) (875) (876) (877) (878) (879) (880) (881) (882) (883) (884) (885) (886) (887) (888) (889) (890) (891) (892) (893) (894) (895) (896) (897) (898) (899) (900) (901) (902) (903) (904) (905) (906) (907) (908) (909) (910) (911) (912) (913) (914) (915) (916) (917) (918) (919) (920) (921) (922) (923) (924) (925) (926) (927) (928) (929) (930) (931) (932) (933) (934) (935) (936) (937) (938) (939) (940) (941) (942) (943) (944) (945) (946) (947) (948) (949) (950) (951) (952) (953) (954) (955) (956) (957) (958) (959) (960) (961) (962) (963) (964) (965) (966) (967) (968) (969) (970) (971) (972) (973) (974) (975) (976) (977) (978) (979) (980) (981) (982) (983) (984) (985) (986) (987) (988) (989) (990) (991) (992) (993) (994) (995) (996) (997) (998) (999) (1000)	7

電子科学 (演習シリーズ)

巻
号
目
次

121. バルス回路設計演習	1
122. バルス回路設計演習	14
123. バルス・デジタル回路の設計演習	13
124. デジタル回路設計演習	6
125. 四則演算回路の設計演習	11
126. デジタルICの基本演習	10
127. IC機器の設計演習	9
128. 電子計算機演習	3
129. マイコン・プログラムの演習	8
130. ALGOLの演習	2
131. COBOLの演習	4
132. FORTRANの演習	5
133. プログラムの演習	7
134. シミュレーションの演習	12